mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

TOMO 8

Editorial Delta, S.A.


mi COMPUTER

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, S. Tarditti, F. Martín
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

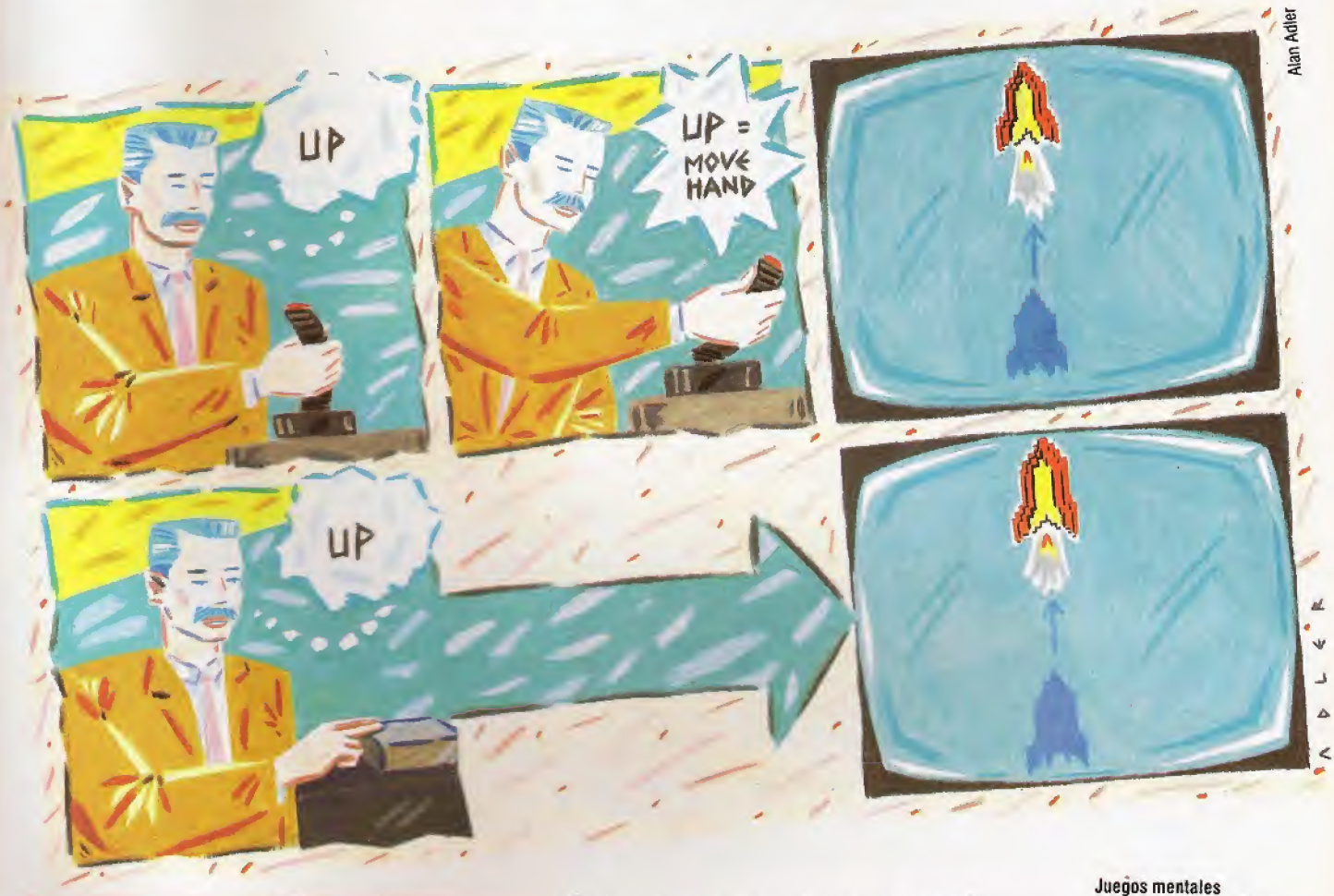
ISBN: 84-7598-152-6 (tomo 8)
Depósito legal: B. 52/1984
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona)
Impreso en España - Printed in Spain - 1985

mi COMPUTER

VOLUMEN 8

Editorial Delta, S.A.

Dominar la materia



Alan Adler

Juegos mentales
El concepto de *mindlink* o conexión mental suprime la etapa mecánica intermedia de traducir los impulsos del jugador en señales comprensibles para el ordenador. Al permitir una relación más directa entre usuario y máquina, la *mindlink* ofrece un software más amable y una entrada más rápida

Controlar un ordenador por medio de la mente ya no es sólo una idea fantástica creada por la ciencia-ficción, sino una palpable realidad

Desde la época de los primeros ordenadores el teclado ha sido el principal dispositivo de entrada, el medio para introducir información en el ordenador. Probablemente sea el vehículo más eficaz para entrar texto y, en menor grado, cifras, al menos hasta que se desarrollen sistemas de reconocimiento de voz para fines generales. Pero, por muy adecuado que sea el teclado para entrar texto, no es necesariamente el mejor dispositivo para entrar otros tipos de información. Si usted desea entrar rápidamente datos de direcciones, como en muchos juegos, es preferible una palanca de mando o un mando de bola y, por lo general, el ratón, el lápiz óptico o la pantalla sensible al tacto representan los mejores medios para entrar datos posicionales, como cuando se realizan selecciones de menús. Un lector de códigos de barras es un método conveniente para entrar largos números en código o seriados.

Todos estos dispositivos de entrada, sin embargo, adolecen de similar inconveniente: son indirectos. Aunque una palanca de mando es más adecuada que un teclado para practicar juegos (p. ej., para desplazar la nave espacial hacia arriba, uno empuja hacia adelante el bastón de la palanca de mando), aún representa una interrupción, un paso entre lo que uno desea que suceda en el ordenador y lo que realmente sucede. Uno piensa "hacia arriba", traduce mentalmente esta idea en "palanca de mando hacia adelante" y luego, físicamente, la empuja. Para una interacción más rápida y más directa entre el usuario y el ordenador necesitamos eliminar este paso intermedio: ¿por qué no sólo pensar "hacia arriba" y lograr que el ordenador responda directamente al pensamiento?

Imagínese lo que sería jugar al *Defender* simplemente pensando "arriba", "abajo", "girar", "disparar", etc., o escribir una carta pensando sólo en las palabras. El procesador de textos telepático está aún a algunos años de distancia, pero el juego *Defender* controlado mediante el pensamiento es en la actualidad una realidad, hecha posible en virtud de un concepto que se conoce como *mindlink* o conexión mental.

En el concepto de *conexión mental*, el pensamiento (o, con mayor precisión, los cambios fisioló-

Notas mentales

En la fotografía vemos una demostración de un prototipo del sistema GSR operando con un ordenador Apple. Se han desarrollado sistemas similares para ser utilizados con los PC IBM y Commodore, pero es probable que el desarrollo de software eficaz lleve aún algún tiempo. Roger Dilts, presidente de Behavioral Engineering y autor de software GSR, está interesado en combinar *mindlink* con NLP (*neuro-linguistic programming*: programación neurolingüística), rama de la psicología que ha estado en gran parte dedicada al estudio del aprendizaje. Mediante el empleo de la técnica GSR para controlar el estado mental del usuario, el software puede ir comprobando el grado de tensión de éste e ir regulando el flujo del programa. Roger Dilts piensa que los programas educativos, por ejemplo, podrán evaluar la respuesta emocional del estudiante ante el material presentado. Si la presentación es demasiado compleja, el ordenador captará la tensión emocional y permitirá cambiar el ritmo del programa o simplificar el tema de estudio

gicos que se producen a consecuencia de los cambios en los patrones del pensamiento) se utiliza para controlar un dispositivo electrónico. Para el ordenador este dispositivo es como una interface, de forma muy similar a una palanca de mando o cualquier otro dispositivo de entrada. La conexión mental se basa en lo que se conoce como "fenómeno GSR", en función del cual los cambios en el estado emocional del individuo se manifiestan en la conductividad eléctrica de la piel. El usuario se conecta mediante electrodos a un medidor de resistencia. Las señales de este medidor se envían a la puerta para el usuario de un ordenador y son interpretadas y obedecidas mediante un software escrito especialmente. La empresa Behavioral Engineering, con sede en California, ha basado en esta técnica tanto juegos como software práctico, y entre sus productos en este campo hay una versión simplificada del *Defender*. Se sabe, asimismo, que otras importantes empresas de ordenadores han estado trabajando en este campo durante los últimos años, en especial Atari, pero sólo recientemente han obtenido cierto nivel de éxito.

En el *Defender* convencional el jugador controla una nave espacial en órbita alrededor de un planeta. El juego consiste en disparar contra las naves de los alienígenas sin que éstas le disparen a uno y sin estrellarse contra el planeta. En la versión de Behavioral Engineering, uno sólo controla la altura de la nave: ¡pero la gran diferencia es que uno la controla a través del pensamiento! La empresa ha diseñado una interface GSR para el Apple IIe. El usuario simplemente coloca los dedos índice y corazón de una mano sobre un dispositivo similar a un ratón, que mide la resistencia a través de los dos dedos y envía los valores resultantes al ordenador. El soft-

Principios del GSR

El concepto de conexión mental se basa en un fenómeno que se conoce bajo tres denominaciones alternativas: GSR (*galvanic skin response*: respuesta galvánica de la piel), PGR (*psychogalvanic reflex*: reflejo psicogalvánico) y EDR (*electrodermal reflex*: reflejo electrodérmico). En este capítulo hemos utilizado el término GSR. Éste alude a cambios en la conductividad eléctrica de la piel que corresponden a cambios en el estado emocional del individuo. La experimentación ha demostrado que, cuanto más tensa está una persona, menor es la resistencia eléctrica de su piel. La aplicación más conocida del GSR son los *polígrafos* o *detectores de mentiras*. Aunque el fenómeno GSR se ha estudiado desde el siglo XIX, se sabe muy poco sobre sus causas. La teoría original afirmaba que el sudor producido por la excitación o la ansiedad actuaba como un conductor electrolítico, disminuyendo, por tanto, la resistencia de la piel. Sin embargo, experimentos más recientes han arrojado dudas sobre esta simplista teoría. No obstante, se sabe que el GSR está directamente relacionado con el grado de tensión existente en el sistema nervioso simpático. Éste, a su vez, depende del sistema nervioso central y, por consiguiente, del cerebro: de allí la posibilidad de controlar un ordenador mediante el pensamiento. Los cambios en la actividad del cerebro producen cambios en el estado del sistema nervioso central; éste produce un cambio en el estado del sistema nervioso simpático, que conduce a cambios en la resistencia de la piel. Y la variación de una corriente eléctrica es la base de toda forma de dispositivo de entrada

Ian Dobbie

sure está diseñado de modo tal que un aumento en la corriente (una resistencia baja originada por una tensión aumentada) produce la elevación de la nave, mientras que una disminución de la resistencia hace que la nave descienda. La idea, por supuesto, no es otra que la de controlar estos movimientos para alinear la nave con los objetivos que se acercan.

El término *conexión mental*, que parece haber sido acuñado por Atari, quizá sea un tanto impreciso, porque la conexión se establece con el sistema nervioso y no con la mente. Ésta es, no obstante, un área oscura: si los dos son interdependientes, ¿tiene sentido separarlos? La mayoría de las personas encuentran que, después de 20 minutos de juego, pueden ejercer un grado de control bastante alto, a menudo sin ser conscientes de la forma en que lo hacen. Algunas personas tensan y relajan su cuerpo ligeramente de forma consciente, mientras que otras simplemente piensan "arriba" o "abajo" y dejan que su sistema nervioso haga el resto.

Además de los juegos, la conexión mental posee aplicaciones más prácticas. Las personas que se hallan totalmente paralizadas conservan aún la capacidad para generar efectos GSR conscientemente, a pesar del hecho de no tener control sobre sus músculos. Ya se ha conectado con éxito un dispositivo GSR al robot Topo a través de un ordenador Apple, permitiendo que una persona paralítica maneje el robot. Otra aplicación interesante se realiza en el espacio, en condiciones de ausencia de gravedad. Sin gravedad para equilibrar la fuerza ejercida, puede ser sumamente difícil operar controles mecánicos. Un GSR podría ser un sustituto ideal para muchos controles mecánicos.

Entre las aplicaciones aún más insólitas se incluye el software perceptor del estado de ánimo. El software educativo, en especial, se podría beneficiar con la realimentación GSR del estado interior del usuario. El individuo podría llevar una muñequera que contuviera los electrodos y dejar que el software calibrara su estado normal relajado. Luego, si en cualquier punto del programa el dispositivo GSR registrara un notable aumento de tensión, se podría suponer que el usuario tenía dificultades y actuar en consecuencia. ¡Hasta el software de gestión podría detectar estrés en el usuario y sugerirle una pausa para tomarse un café!

La rapidez a la que se desarrolle la tecnología GSR dependerá en parte de la introducción de dispositivos de mayor precisión y que respondan con más rapidez, y, en parte también, a nuestra capacidad para perfeccionar nuestra interpretación de los efectos. En cuanto a lo primero, el problema es que, típicamente, la respuesta GSR se produce dos segundos después del acontecimiento y tarda entre diez y diez segundos en desaparecer. Los dispositivos actuales superan hasta cierto punto este inconveniente al medir la velocidad del cambio de la respuesta en lugar de la intensidad de ésta; no obstante, es preciso ir más allá en este sentido. El segundo problema radica en la pobreza de las deducciones que somos capaces de realizar a partir de la realimentación GSR. Sabemos que una disminución rápida de la resistencia de la piel indica algún tipo de estrés, pero todavía no somos capaces de determinar si la tensión es agradable o dolorosa.

Sin embargo, a pesar de los problemas, el GSR ofrece algunas posibilidades apasionantes.

Associated Press

El detector de mentiras

Uno de los principales problemas con que se encuentra la policía y la justicia al realizar una investigación y posterior juicio reside en saber si un sospechoso o un testigo está mintiendo. Si bien un observador experimentado puede detectar mínimos indicios (cambios en la coloración de la piel y la respiración, p. ej.), la medición de estos factores no se ha definido ni catalogado, ni tampoco son admisibles como evidencia. Por este motivo, se ha investigado profundamente para desarrollar un medio objetivo de apreciar la diferencia entre afirmaciones verdaderas y falsas. Uno de los resultados de tal investigación ha sido el *polígrafo* o *detector de mentiras*. En realidad éste no es más que un medidor de resistencia. Se basa en la teoría de que la tensión de un individuo aumenta significativamente al decir una mentira u oír palabras clave relacionadas con el delito, y que este aumento de tensión se refleja en una disminución de la resistencia de la piel. El polígrafo es objeto de gran controversia. Tenido en gran estima por agencias de investigación, en particular en Estados Unidos, sus detractores arguyen que no se sabe lo suficiente acerca del GSR como para interpretarlo de forma fiable, y que diferentes personas pueden reaccionar de forma muy distinta, independientemente de su culpabilidad o inocencia

Para obtener un empleo
En ocasiones se utiliza el GSR para evaluar a los candidatos a un empleo: un detector de mentiras ayuda a calificar las respuestas de un individuo a las preguntas que se le formulan. En la fotografía vemos a un agente de ventas a quien se le están mostrando los resultados de un test que acaba de realizar

Mentiras diáfanas
Un detector de mentiras mide los cambios en el GSR provocados por la respuesta emocional del individuo ante la pregunta formulada. Sin embargo, en la vida real es muy poco probable que un detector de mentiras ofrezca resultados tan evidentes como el que vemos aquí. Las reacciones emocionales varían a tenor de la sensibilidad del individuo ante ciertos temas (p. ej., si el tema le resulta embarazoso) y no dependen sólo de la veracidad o falsedad de las respuestas

Liz Dixon

Estructuras dinámicas

El PASCAL concede una importancia primordial a la precisión de las técnicas de programación al tratar archivos

Ahora podemos seguir desarrollando nuestra base de datos pensando en algunas variables esenciales y en un algoritmo informal.

```
VAR
  lista    : ListaRegistro;
  tamaño : Cardinal;
BEGIN
  tamaño :=0; {tamaño activo de la lista}
  {leer los datos en la lista, actualizando tamaño}
  {clasificar elementos de tamaño en el orden
  correcto}
  {imprimir elementos de tamaño de la lista}
END.
```

Para traducir este algoritmo informal en algo más tangible necesitamos expresar las tres etapas principales del proceso como llamadas a procedimientos (con nombres adecuados) y listar todos los datos conocidos que requerirá cada uno en forma de una lista de parámetros.

Una vez realizado este sencillo paso, podremos escribir todos los bloques de procedimiento como un esqueleto. Por ejemplo, la última sentencia del programa se convertirá en:

```
Imprimir (lista, tamaño)
```

El procedimiento tendrá toda la información que necesita si le pasamos la lista de valores de datos y la cantidad de elementos, de modo que en este caso el encabezamiento no necesitará ninguna VAR:

```
PROCEDURE Imprimir (items  : ListaRegistro;
                    altura : limites);
```

Podríamos seguir adelante pasando a codificar todo el procedimiento, pero por ahora es mejor dejarlo como una "colilla" BEGIN...END y ocuparnos de los otros procedimientos de *nivel 1*. Hemos de elegir un nombre para cada uno, decidir qué elementos de datos es preciso pasar como parámetros a cada procedimiento, y determinar si es necesario pasar algunos de estos elementos como parámetros VAR (dirección).

Todos los métodos de estructuración de datos que hemos analizado hasta ahora han sido de tamaño fijo. Ello se especifica en las definiciones TYPE y, por consiguiente, se conoce en tiempo de compilación. El PASCAL proporciona estructuras de datos avanzadas cuyo tamaño puede variar (e incluso también su tipo, con ciertas restricciones) durante la ejecución de un programa. El tamaño de una estructura avanzada sólo está limitado por la memoria física o el almacenamiento de apoyo disponibles, y la estructura avanzada más familiar es el archivo secuencial.

Al igual que una matriz, cada elemento de un archivo puede ser de cualquier tipo, simple o estructurado, con la excepción de que uno no puede tener un archivo de archivos. Utilizando nuestra anterior definición de tipo de registro, podríamos añadir las siguientes declaraciones:

```
TYPE
    TipoArchivo=FILE OF datos;
VAR
    ArchivoDatos:TipoArchivo;
```

que nos permitirían crear y procesar un archivo que contuviera un número infinito de componentes, siendo cada uno de ellos un registro de un nombre, deuda y cualquier otro campo que deseáramos. Igual que decimos read (simbolo), queriendo significar leer un único char de la entrada del archivo, podemos decir: read (ArchivoDatos,item) o write (ArchivoDatos,item) y manipular toda una estructura de registro como un objeto de un solo dato. Si estos archivos sólo se requieren durante la ejecución del programa, la otra única exigencia consiste en abrirlos para leerlos o escribir en ellos mediante los procedimientos predefinidos reset y rewrite, respectivamente.

Si usted desea darles un carácter permanente, lo cual es más probable, entonces los identificadores de archivo deben especificarse en la lista de parámetros del encabezamiento del programa. En este caso, por ejemplo:

```
PROGRAM ManipuladorDatos (input,output,ArchivoDatos);
```

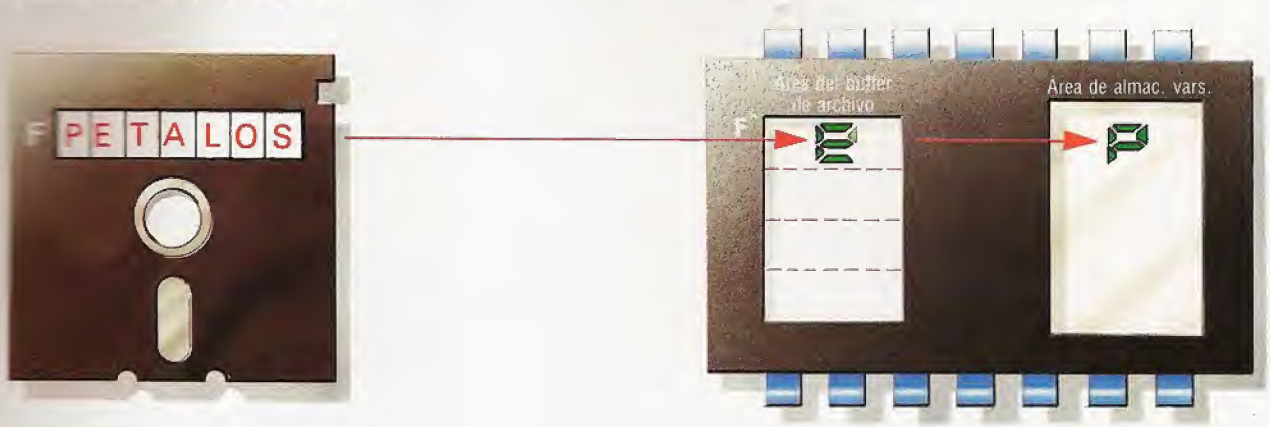
Cada vez que utilizamos sentencias read o write, llamamos a procedimientos de E/S de archivos predefinidos del PASCAL. Los dos únicos archivos que, en realidad, conocemos hasta ahora son los dispositivos de E/S estándares de nuestro ordenador. El archivo input normalmente es un teclado y output será, invariablemente, una pantalla VDU en los sistemas de microordenador. Al simular que estos dispositivos son archivos, en PASCAL la manipulación de todas las E/S resulta sumamente coherente. Recuerde que cuando decimos write(N), omitiendo cualquier nombre de archivo, el valor de N se imprime en forma de caracteres en el archivo output. Al omitir un nombre de archivo en sentencias read o ReadLn, se pasa por defecto al archivo input. Estos dos archivos son archivos de texto, es decir, cada "registro" es un valor de un único caracter.

Archivos de texto

Sin embargo, a diferencia de otros archivos, los archivos de texto pueden tener (además de una marca de final de archivo) indicadores especiales de final de línea empotrados en cualquier punto dentro de ellos.

Estos indicadores varían con los diferentes sistemas operativos, y pueden constar de un solo carácter de control, de dos caracteres (CR y LF, p. ej.), o

Leyendo de un archivo



tal vez de ningún carácter, almacenándose, en cambio, la longitud de cada línea.

Con el fin de preservar la portabilidad, el PASCAL proporciona la función EoLn (F) y los dos procedimientos predefinidos ReadLn(F) y WriteLn(F), los cuales sólo se pueden emplear con archivos de tipo text. La función EoF(F), por supuesto, se puede utilizar con cualquier tipo de archivo. Dado que el indicador de final de línea puede o no ser un único carácter, la lectura de un valor char cuando EoLn es verdadera devolverá un carácter Space. Por consiguiente, normalmente debemos comparar previamente con EoLn. Puesto que tanto input como output existen antes y después de la ejecución de cualquier programa, están permanentemente abiertos, y no es preciso cargarlos ni crearlos de forma explícita.

Con archivos distintos de input y output, debemos asignarlos a un nombre del sistema externo y luego abrirlos para la lectura con una llamada al procedimiento reset (p. ej., reset (AlgunArchivo)), o bien crear una entrada en el directorio preparatoria para escribir en ellos mediante rewrite (Otro Archivo). Por lo tanto, un esquema general para procesar un archivo de texto es simplemente:

```
{abrir el archivo}
WHILE {no se llegue al final del archivo fuente} DO
  WHILE {no se esté en el final de una línea} DO
    {leer un carácter}
    {procesar el carácter}
    {saltarse el final de la línea}
```

PUT y GET

Los procedimientos read y write en realidad se implementan utilizando buffers de archivos y las primitivas de E/S put (para salida) y get (para entrada desde archivos). Cuando se reescribe un archivo, en el buffer no se coloca ninguna información hasta que se lleva a cabo un write. Éste en realidad consta de las dos operaciones:

```
F^ := datos;
put (F)
```

Del mismo modo, la sentencia read (F,datos) también se puede expresar:

```
datos := F^
get (F)
```

Por lo tanto, las sentencias de E/S de nuestro procedimiento Copiar:

```
read (fuente,caracter);
write (destino, caracter)
```

se podrían haber codificado igualmente como:

```
destino^:=fuente^;
put (destino);
get (fuente)
```

De esta manera no necesitaríamos de la variable char local, caracter. Tal vez se podría alcanzar una mejor comprensión de la operación de ReadLn (F) si la expresáramos en términos de estas primitivas:

```
WHILE NOT EoLn (F) DO
  get (F); {descartar el resto de la línea, si lo
  hubiera}

get (F) {.. y saltar el/los caracter/es EoLn}
```

De modo, entonces, que tras una ReadLn el buffer del archivo siempre contendrá el primer elemento de la siguiente línea a leer. Éste podría ser un espacio si EoLn (F) fuera verdadera, o estar indefinido si EoF (F) fuera verdadera. Obviamente, es ilegal intentar leer un archivo cuando EoF es verdadera, pero también es un error llevar a cabo cualquier operación, incluyendo la comprobación del buffer del archivo. Ello significa que se debe ser cuidadoso al manejar archivos que no sean ni input ni output. Pero, al mismo tiempo, el tener que estar atento a condiciones de error potenciales tales como éstas favorecerá la escritura de un software eficaz.

Ahora que sabemos cómo "anticiparnos" a un flujo de datos venidero, podemos formular el procedimiento SaltarBlancos que proponíamos en el capítulo anterior.

```
PROCEDURE SaltarBlancos (VAR F : text);
  CONST
    espacio = ' ';
  VAR
    hecho : boolean;
  BEGIN
    hecho := EoF (F);
    IF NOT hecho THEN
      hecho := input^ > espacio;
```

A la sombra de un archivo
El proceso de abrir un archivo F en PASCAL prepara una zona buffer asociada, F^, en la cual se lee el primer carácter del archivo. Cuando se realiza una lectura, el carácter de la zona del buffer se le asigna a una variable y se transfiere a la zona del buffer el siguiente carácter del archivo. De este modo, si los primeros caracteres de F fueran PETALOS, al abrir el archivo P se leería en F^. Tras la primera operación de lectura, P sería asignada a una variable del PASCAL, y sería reemplazada por E en el buffer, F^

```
    WHILE NOT hecho DO
      BEGIN
        get (F);
        hecho := EoF (F);
        IF NOT hecho THEN
          hecho := F^ > espacio
      END
  END; {SaltarBlancos}
```

Observe que así saltaremos todos los espacios en blanco de un archivo de texto, incluyendo los caracteres de final de línea. Si sólo deseáramos saltar espacios en una línea dada, podríamos alterar la asignación condicionada:

```
hecho := EoLn (F) OR (F^ > espacio)
```

Si posteriormente deseáramos saltar todos los espacios en blanco:

```
REPEAT
  SaltarBlancos (F);
  IF NOT EoF (F) THEN
    TextoHallado := NOT EoLn (F)
  UNTIL TextoHallado OR EoF (F)
```

Con esta última modificación, resulta muy sencillo escribir programas interactivos que comprueben entradas nulas.

Por ejemplo:

```
REPEAT
  write ('Entre datos:');
  SaltarBlancos (input)
UNTIL NOT EoLn (input)
```

Esto fracasará en el caso de que se entre el carácter de control que utilice el sistema para indicar el final del archivo, pero siempre podríamos usar el esquema anterior si deseáramos conseguir que nuestro programa fuera absolutamente seguro.

El procedimiento assign ha llegado a considerarse como una *ampliación estándar* y podría ser que pronto se adoptara con carácter oficial, así como open y seek para archivos de acceso directo. Otras ampliaciones esenciales de uso común son:

```
FUNCTION Fstat (NombreArchivo)
```

que devuelve un resultado booleano: (true si ya existe el archivo), y una facilidad:

```
PROCEDURE Rename (NombreViejo, NombreNuevo)
```

Complementos al PASCAL

No existe restricción alguna sobre el número de archivos que es posible tener abiertos. Sin embargo, puesto que éstos requieren la utilización del OS, nos hallamos en un campo en el cual existen divergencias. Por ejemplo, algunas versiones del lenguaje no soportan archivos. Las convenciones del OS para la asignación de nombres a los archivos puede, por tanto, suponer un problema. Muchos PASCAL de ordenadores centrales toman los primeros 10 caracteres, más o menos, de todo identificador de archivo, y lo relacionan con un archivo que posea ese nombre, de modo que nuestro ejemplo de archivo permanente crearía uno llamado ARCHIVODATOS. En muchos sistemas, las convenciones para los nombres son tales que esto no es posible. Ejemplos: A CP/M-FIL.DAT, #4:APPLEFOR MAT, etc. La "ampliación estándar" para conectar un identificador de archivo es el procedimiento de asignación, empleado antes de reestablecer o reescribir:

```
assign (IdentificadorArchivo,SerieNombre)
```

De modo que, a nuestros fines actuales, bastaría assign (ArchivoDatos,D:NuestroArch.dat)

Programa CopiarTexto

El PASCAL proporciona el identificador "text" para el tipo más común de archivo, y éste se utiliza cuando declaramos todas las variables del archivo de texto en la lista de parámetros del encabezamiento del programa. Aquí ofrecemos una facilidad para copiar archivos de texto. Con el objeto de lograr que el programa sea útil con el carácter más general posible, hemos formulado todo el proceso de copiado como un procedimiento separado denominado Copiar, que puede procesar dos archivos de texto cualesquiera. Recuerde que los compiladores no estándares exigen la utilización de reset (F1,'Fuente'), etc., en el programa principal. Observe que ambos archivos se pasan al procedimiento Copiar como parámetros VAR. Ello se debe a que no sólo se actualiza el archivo de destino, sino que se lee el archivo fuente y, por lo tanto, cambia el estado del archivo. Por este motivo, las variables de archivo siempre se pasan por dirección, jamás por valor. Aparte de cualquier otra consideración, un parámetro de valor implica una copia local. Pasar grandes estructuras, tales como matrices de registros, también se puede considerar una excepción por razones de conservación de memoria

```
PROGRAM     CopiarTexto     (F1,F2);

VAR
    F1,
    F2          : text;

{11111111111111111111111111111111}

PROCEDURE   Copiar (VAR  fuente,
                         destino   : text);
  VAR
    caracter              : char;
  BEGIN
    WHILE NOT EoF (fuente) DO
      BEGIN {copiar una linea:}
        WHILE NOT EoLn (fuente) DO
          BEGIN
            read (fuente, caracter);
            write (destino, caracter)
          END;
            {ahora copiar el final de linea:}
        ReadLn (fuente);
        WriteLn (destino)
      END
  END;    {Copiar}
{11111111111111111111111111111111}

BEGIN     {CopiarTexto — Programa principal}

  assign (F1, 'Fuente');
  reset (F1);                 {localizar y abrir para lectura}
  assign (F2, 'Destino');
  rewrite (F2);               {crearlo}
  Copiar (F1, F2)

END.
```

Administrador eficaz

En esta ocasión concentraremos nuestra atención en el "MicroPen", DBM diseñado para el Amstrad CPC 464

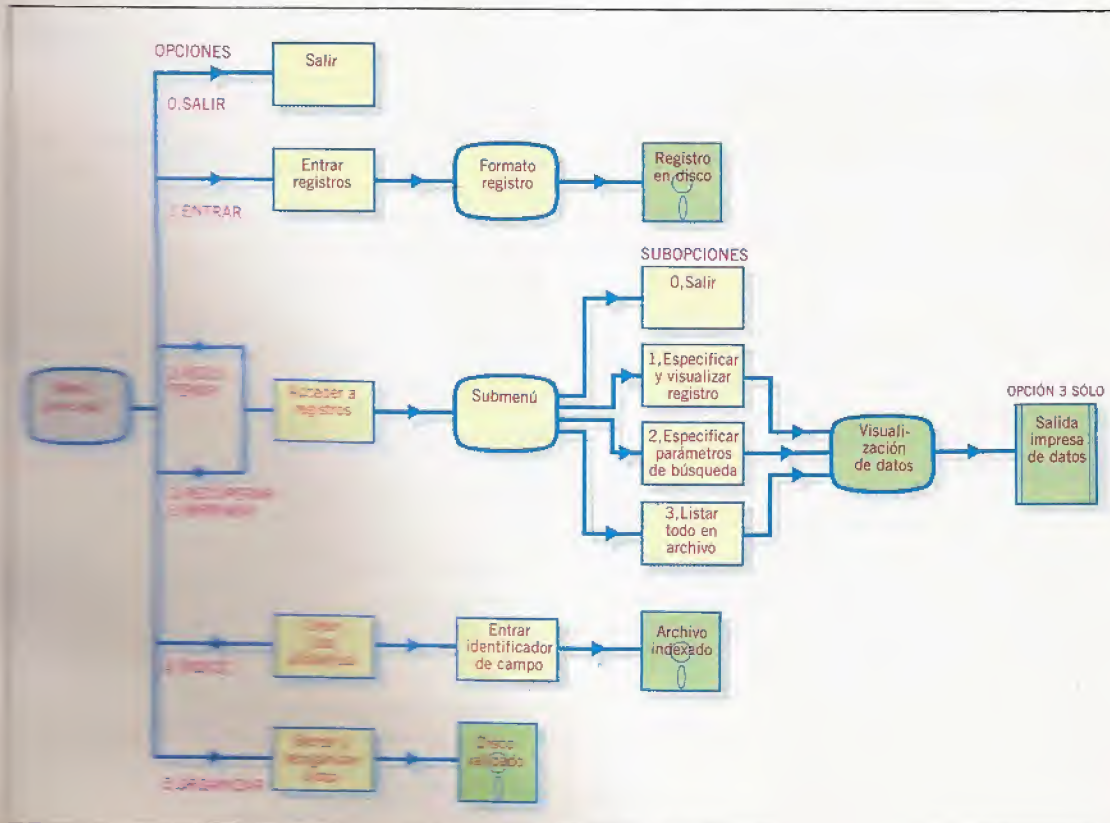
Descrito como un "sistema de archivo de bases de datos para el CPC 464", *MicroPen* está editado por la división Amsoft de Amstrad, pero es obra de la empresa de software Intelligence Ireland. *MicroPen* forma parte de un trío de programas que comprende un DBM, un procesador de textos y una hoja electrónica. Desde el punto de vista conceptual, el paquete guarda similitud con juegos como el *Lotus 1-2-3* y el *PIPS* de Sord. El componente de DBM incluye un procesador de textos denominado Penform. El mismo se utiliza para crear formatos en pantalla de los registros a utilizar dentro de un archivo DBM. El DBM propiamente dicho se denomina simplemente Pen.

La versión Amstrad del *MicroPen* funciona bajo CP/M-80 y, por consiguiente, requiere al menos una unidad de disco DDI-1. Aunque esto parece un gasto adicional algo desafortunado, usted comprenderá enseguida que la capacidad y velocidad de los discos es esencial; los DBM basados en cassette pueden resultar penosamente lentos y frustrantes.

Para crear un archivo de base de datos es necesario crear primero el trazado o formato para los registros que compondrán el archivo. Ello se realiza ejecutando el Penform. Si bien en la documentación se alude a éste como un procesador de textos, en realidad no es más que un editor en pantalla. Es decir, permite entrar o modificar caracteres en la pantalla, desplazando el cursor mediante las teclas para el mismo. Las teclas del cursor se complementan con instrucciones de edición simples, tales como [CTRL] Y para suprimir una línea o [ESC] E para salvar en disco. El Penform no es un auténtico procesador de textos, puesto que no se puede utilizar para crear y editar documentos completos como cartas o artículos.

Hay algunas limitaciones en cuanto a la cantidad de información que se puede incluir en un archivo de base de datos *MicroPen*. Un registro (denominado *layout* —trazado— en la documentación) puede tener hasta 100 campos, pero ningún registro puede contener más de 1 024 caracteres. Teóricamente, un archivo puede contener hasta 32 750 registros, pero en la práctica no es viable una cantidad tan alta de registros en un sistema basado en disco flexible. Los 32 750 registros completos, conteniendo cada uno de ellos los 1 024 caracteres admisibles como máximo, requerirían más de 33 megabytes de almacenamiento en disco sólo para los datos en bruto (suponiendo que no se utilicen técnicas para compresión de datos).

Una característica inusual del *MicroPen* es que a cada uno de los campos de un registro se le ha de otorgar un identificador exclusivo en pantalla.



Plan de acción
Éstas son las diferentes opciones y procedimientos de operador que ofrece el menú principal del *MicroPen*. Los DBM que poseen su propio lenguaje de programación incorporado, como el *dBase II*, permiten la construcción de secuencias de procedimiento similares según las propias necesidades

Estos identificadores se denominan *señaladores de campo* y pueden ser cualquier carácter imprimible a excepción del corchete, que no está disponible porque posee un papel especial como delimitador de campo.

Si estuviéramos utilizando el *MicroPen* como DBM para nuestra base de datos COMPONENTES, un trazado de registro creado por Penform tendría el siguiente aspecto:

```
Número de componente del fabricante:  [A        ]
Número de componente nuestro:  [B               ]
Precio:  [C   ] Cantidad en stock:  [D           ]
Descripción:  [E                                ]
Proveedor:  [F                                  ]
Num. teléfono del proveedor:  [G                ]
Hablar con:  [H                                 ]
```

Tras haber creado un formato como éste, es guardado en disco empleando un nombre de archivo como COMPONENTES.INP (la "extensión" .INP del nombre del archivo la exige el programa Pen DBM).

Ejecución del "Pen"

Al ejecutarlo, el *Pen* visualiza un mensaje en la pantalla que pregunta qué archivo de base de datos se ha de utilizar. Para emplear nuestra base de datos de inventario, responderíamos COMPONENTES (no se requiere aquí la extensión .INP para el nombre del archivo). *Pen* visualiza entonces un menú, conocido como *menú principal*, que ofrece seis opciones:

```
Base de datos: COMPONENTES, por registro 328, registros en archivo 0.

0=Salir, 1=Entrar, 2=Recuperar, 3=Recuperar e imprimir, 4=Indice, 5=Organizar
```

Para entrar registros se necesita la opción 1. La línea superior (línea de estadística) señala que hay 328 caracteres por registro, pero que en el archivo no hay (todavía) ningún registro. Tras la selección de la opción 1 se visualiza en la pantalla el formato de registro con instrucciones. En nuestro ejemplo, en la pantalla se visualizaría:

```
Pulsar {ESC} tras completar entrada de datos para registro 1
Pulsar^C para borrar campo

Número de componente del fabricante:
Número de componente nuestro:
Precio:                          Cantidad en stock:
Descripción:
Proveedor:
Num. teléfono del proveedor:
Hablar con:
```

Los datos se digitan de la forma habitual. Tras entrar los datos para un campo, la pulsación de ENTER concluye la entrada para ese campo y desplaza el cursor hasta el campo siguiente. Los errores cometidos en un campo se pueden corregir mediante el empleo de la tecla Delete. Una vez entrado correctamente todo el registro, utilizando la tecla Escape se visualizará un submenú:

```
0=Salir
1=Continuar
2=Escribir registro en archivo
```

La opción 2 escribe el registro en disco y visualiza el formato para el siguiente registro.

La opción 2 del menú principal permite acceder a registros o recuperarlos y da lugar a otro submenú:

```
Recuperar por:  0=Salir
                1=Número de registro
                2=Búsqueda
                3=Listar todo el archivo
```

Las opciones 0 y 1 se explican bastante por sí solas; la opción 1 permite especificar y visualizar un registro; da por sentado, no obstante, que usted conoce el número de registro que quiere, aunque es poco probable que usted lo recuerde si el archivo contiene muchos registros. La opción 2 permite especificar varios parámetros de búsqueda para poder localizar cierto registro que cumpla la especificación. El submenú de búsqueda es:

```
Pulsar Escape cuando el perfil de búsqueda esté completo. Para establecer modalidad de búsqueda:
^Q=Contiene, ^W=No contiene, ^E=Igual a, ^R=No igual a, ^T=Mayor que, ^Y=Menor que
```

Usted ya habrá percibido que hay una gran incoherencia en cuanto a las instrucciones y opciones de menú para las diversas partes del paquete: algunas veces debe pulsarse la tecla Escape, otras veces una combinación de CRTL–letra, otras un número de menú. Muy raramente las combinaciones CTRL–letra poseen algún verdadero valor mnemónico: ^E para hallar un registro con campos de comparación durante una búsqueda, ^Q para especificar Contiene, etc. Sea como fuere, las opciones de "búsqueda" permiten una razonable flexibilidad para especificar los registros que se han de buscar.

Habiendo dado los parámetros que definen el registro requerido, las combinaciones CTRL–letra permiten localizar registros que contengan los datos especificados, que no contengan los datos especificados, que posean campos con datos emparejados, que no posean campos con datos emparejados, o que posean un valor de campo menor que el valor de campo especificado. Si quisiera, por ejemplo, localizar registros de componentes con un PRECIO inferior a 37,50, lo podría conseguir con gran facilidad.

MicroPen puede crear un índice de todas las entradas de un archivo de base de datos. La indexación se efectúa por campo, y debe conocerse también la cantidad máxima de registros. El campo se especifica por su identificador, no por el nombre que le hemos dado al campo. Para obtener un índice de Número de componente nuestro, y suponiendo que en el archivo hubiera 500 registros, especificaríamos B=#500 (siendo B el identificador para nuestro campo Número de componente nuestro).

MicroPen ofrece capacidades de búsqueda e indexación moderadamente avanzadas, y por su precio es asequible a muchos usuarios de ordenadores personales. No ofrece, sin embargo, la ventaja de un lenguaje de programación incorporado, como los que incluyen *Archive* y *dBase II*. Aunque este último es un paquete caro diseñado para ordenadores caros, *Archive* viene "gratis" con el Sinclair QL y a un precio global comparable al de un Amstrad CPC 464 más unidad de disco y software *MicroPen*. Ésta es la clase de consideraciones a tener presentes antes de adquirir un sistema de ordenador con el software de DBM asociado.

Ampliación Amstrad

El paquete de unidad de disco DDI-1 de Amstrad proporciona una mejora a buen precio para la máquina

El concepto del ordenador como un artículo más de la electrónica doméstica, como un televisor o un equipo de alta fidelidad, instalado con carácter permanente en un rincón de la habitación en vez de tenerlo que montar y conectar a otros componentes cada vez que se desea utilizarlo, fue obviamente recibido con satisfacción por los entusiastas. Existe, asimismo, la ventaja de evitar conflictos con quienes desean usar el televisor.

El Amstrad es muy popular entre los usuarios personales, pero posee un inconveniente fundamental: a pesar de que la empresa incorporó en la máquina una unidad de cassette, el ordenador carecía de una capacidad de almacenamiento rápido. En el momento del lanzamiento se prometió una unidad de disco, lo que llevó a mucha gente a adquirir el ordenador a la espera de que la misma saliera rápidamente a la venta. Sin embargo, las remesas no aparecieron hasta el año siguiente y desde entonces la adquisición del dispositivo se puede realizar cada vez con más facilidad a través de las tiendas minoristas.

El paquete de unidad de disco se compone de la unidad de disco, una interface (que permite conectar la máquina a la puerta para disco flexible de la parte posterior del ordenador), un disco de sistema y un manual. El otro extremo del cable de la interface se desliza en el conector de 34 vías de la parte posterior de la unidad de disco.

La unidad de disco propiamente dicha utiliza discos de 3 pulgadas de estándar Hitachi. Ésta parece una elección extraña, ya que es Sony y no Hitachi quien aparece como probable ganadora de la batalla por captar el mercado de microflexibles. El formato Sony de 3 ½ pulgadas parece estar a punto de convertirse en el estándar, puesto que un creciente número de fabricantes, incluyendo Apple, Apricot y, más recientemente, Acorn con su Electron, han adoptado los discos Sony para sus unidades. A excepción de Amstrad, ningún otro de los grandes fabricantes de ordenadores parece haber optado por los discos Hitachi.

En el interior de la unidad, dos motores claramente visibles controlan la rotación del disco y el cabezal de lectura/escritura. En la parte posterior está la fuente de alimentación, separada del mecanismo de la unidad de disco mediante una placa metálica que la protege del exceso de calor y los campos magnéticos. En la parte posterior de la carcasa y encima de la puerta para la interface hay un interruptor de on/off.

Los discos utilizados en el Amstrad son similares, en concepto, a sus equivalentes Sony, aunque su aspecto es muy diferente. Los discos Amstrad miden 100x80x4 mm y, al igual que los discos Sony, se hallan dentro de una carcasa plástica y poseen una placa metálica sobre la ventana de lectura/escritura que protege al disco de las huellas dactilares o la suciedad. Ésta se retrae cuando el disco se coloca en la unidad. Los discos Amstrad, sin embargo, poseen sus escudos en el interior de la carcasa metálica, mientras que Sony los ha colocado en el exterior.

Chris Stevens

Escalando el mercado
La unidad de disco Amstrad DDI-1 permite que los usuarios amplíen sus sistemas colocándolos a la altura de las especificaciones que requiere un microordenador "serio". Con la adición de los tres programas que se suministran junto con la unidad de disco (CP/M, AmsDOS y Dr LOGO), el Amstrad CPC 464 posee ahora un número mucho mayor de aplicaciones. La unidad se conecta al ordenador mediante el cable de interface que se proporciona, a través de la puerta interface para disco flexible

La unidad operó de forma rápida y fiable y no tuvimos ningún problema para hallar archivos y cargarlos en unos pocos segundos, pero pareció más ruidosa que el producto de Sony. No obstante, resultó más silenciosa que la media de unidades de disco flexible de 5 ¼ pulgadas promedio.

El disco del sistema

El disco del sistema contiene tres utilidades básicas: AmsDOS, el propio sistema operativo de disco de Amstrad, y dos utilidades de Digital Research: el sistema operativo de disco CP/M, tan ampliamente utilizado, y Dr LOGO, una popular implementación del lenguaje de aprendizaje y de gráficos tortuga que cada vez se está extendiendo más, con evidencias incluso de desplazar al BASIC como lenguaje principal en los ordenadores personales.

Refiriéndonos en primer lugar al AmsDOS, quizá ésta sea la más floja de las tres utilidades proporcionadas. Para poder ejecutar el AmsDOS, debe primero cargarse el CP/M. Esto tal vez parezca extraño, pero el resultado es que el AmsDOS utiliza igual memoria que el CP/M solo y, por tanto, el CP/M se desecha tras haber cargado al AmsDOS. A diferencia del CP/M, que es un siste-

ma operativo autocontenido, el AmsDOS simplemente añade instrucciones de operación de disco al carácter barra vertical | (Shift @); por ejemplo: | DRIVE y | DIR.

El procedimiento para manipular archivos en disco desde AmsDOS es uno de los más peculiares que puedan verse en cualquier micro personal. Por ejemplo, para borrar (ERA*se*) un archivo, usted debe primero asignar el nombre del archivo a una serie; sólo entonces podrá suprimir el archivo mediante el borrado de la serie. Por consiguiente, para borrar el archivo FACTURA.$$$, la secuencia de instrucciones sería:

```
A$="FACTURA.$$$"
| ERA,aA$
```

Este método se complica aún más al renombrar un archivo (REN*ame*), dado que dos archivos diferentes (nombre viejo y nombre nuevo) se han de asignar a series que se puedan manipular. A muchos usuarios les resultará más sencillo cargar el CP/M para tales procedimientos, que se pueden llevar a cabo mediante una única línea. No obstante, el CP/M posee sus propias dificultades. Fundamentalmente, no hay facilidades para ejecutar el BASIC Amstrad bajo CP/M, de modo que usted se verá obligado a utilizar el AmsDOS si desea programar el sistema de unidad de disco en BASIC.

El sistema operativo CP/M del disco de sistema es la versión 2.2 que se ha venido aplicando en gran número de máquinas de gestión de ocho bits en la última década. Junto con el propio CP/M se proporcionan en el sistema las instrucciones "transitorias" usuales. Las instrucciones transitorias son aquellas que están retenidas permanentemente en disco y que se cargan en la memoria del ordenador sólo cuando se las necesita; luego son desechadas por el sistema operativo CP/M. Aquí el problema es que para poder utilizar una gran cantidad de instrucciones en CP/M (como PIP, que transfiere un archivo de un dispositivo periférico a otro), el sistema operativo realmente requiere que haya dos discos instalados en el ordenador, uno para retener el disco de sistema de modo que se pueda acceder rápida y sencillamente a las instrucciones transitorias, y otro para retener los archivos.

Aparte de la necesidad de utilizar el CP/M para intercambiar continuamente los discos de datos y de sistema, poniéndolos y sacándolos en una única unidad, muchas instrucciones no tienen en cuenta que el usuario tenga una sola unidad de disco disponible y esperan que pueda transferir los archivos de una unidad a otra. Para contrarrestar esto, Amstrad ha incluido instrucciones transitorias adicionales para la transferencia de archivos con una sola unidad. FILECOPY proporciona mensajes que permiten el intercambio de discos al transferir un archivo y, de forma similar, DISCCOPY copia un disco completo. Aun así, usted habrá de adquirir dos unidades para sacar el máximo partido del sistema.

Puede decirse, no sin cinismo, que la de Amstrad es una maniobra inteligente para obligar a los clientes a adquirir dos unidades de disco en lugar de una. Sin embargo, una explicación más benévola es que el suministrar el CP/M tiene su sentido en función de la estrategia a largo plazo de Amstrad. La filosofía que subyace tras la gama de ordenadores de la empresa no es la de crear productos de tecnología punta, sino la de proporcionar equipos ya probados y comprobados que se puedan utilizar para una amplia gama de aplicaciones. De allí la decisión de la empresa de optar por el procesador Z80, cuando la mayoría de los fabricantes se disputan por producir máquinas de 16 bits. El empleo del CP/M encaja bien, puesto que, aunque no sea el sistema operativo de disco más amable con el usuario, es adaptable y en todo el mundo hay miles de programadores que poseen la experiencia necesaria para escribir software para ejecutar bajo el sistema.

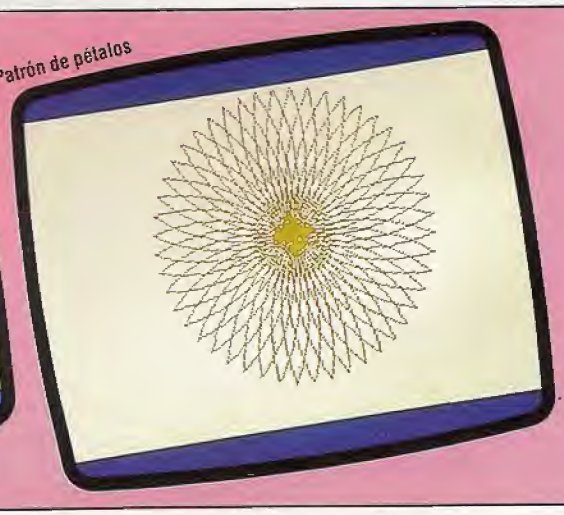
A pesar de los problemas que supone la utilización del CP/M con una sola unidad de disco disponible, Amstrad merece felicitaciones por proporcionar una implementación completa de un OS de disco tan potente como éste en un micro personal.

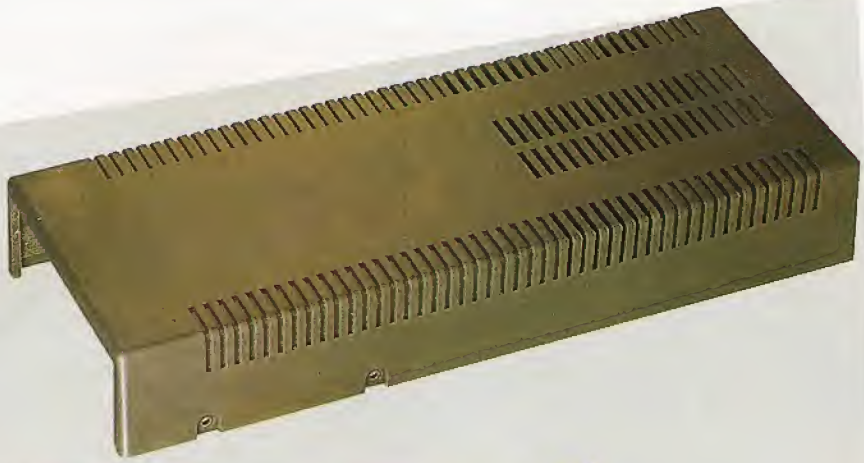
El LOGO de Digital Research

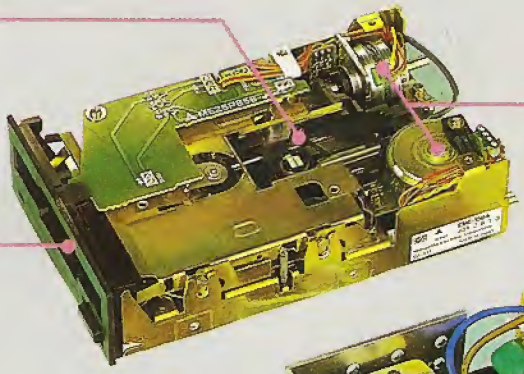
Dr LOGO es una versión del popular lenguaje que se utiliza en muchos centros educativos europeos. El lenguaje opera bajo CP/M, que, por consiguiente, se ha de cargar antes de poder cargar el LOGO, y el Dr LOGO puede aprovechar las facilidades operativas de disco ampliadas del CP/M. Por ejemplo, el CP/M permite que los archivos se puedan llamar sin tener que digitar un nombre completo; de modo que, digitando un nombre parcial, el lenguaje llamará a todos los archivos que encajen con esa descripción parcial.

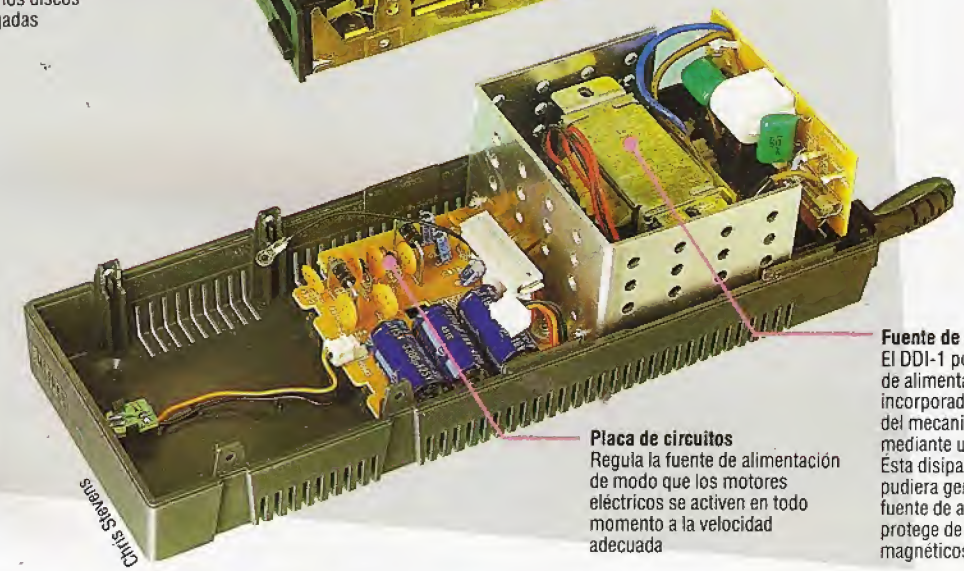
Arte de tortuga

Estos patrones se crearon utilizando el lenguaje Dr LOGO. Los procedimientos que empleamos para crearlos sólo exigieron unas pocas líneas de código cada uno. Aunque el Dr LOGO permite alterar los colores de fondo y primer plano desde el lenguaje, no se ha previsto la modificación de la forma de la tortuga ni el coloreado de las formas que se dibujan. No obstante, desde el LOGO se puede generar sonido









Cabezal de lectura/escritura
El cabezal de lectura/escritura detecta los cambios que se producen en el campo magnético del disco y los traduce a señales eléctricas

Ranura para disco
Aquí se insertan los discos Hitachi de 3 pulgadas

Motores
Estos motores se utilizan para alimentar la unidad. El motor de abajo hace girar el disco, mientras que el otro mueve el cabezal de lectura/escritura a la pista correcta

Fuente de alimentación
El DDI-1 posee su propia fuente de alimentación eléctrica incorporada. Está separada del mecanismo de la unidad mediante una placa metálica. Esta disipa el calor que pudiera generarse dentro de la fuente de alimentación y la protege de los campos magnéticos

Placa de circuitos
Regula la fuente de alimentación de modo que los motores eléctricos se activen en todo momento a la velocidad adecuada

Chris Stevens

AMSTRAD DDI-1

DIMENSIONES
280×105×75 mm

CAPACIDAD
Disco de sistema: 169 K; disco de datos: 178 K

INTERFACES
Interface individual en paralelo de 34 vías. El cable de interface tiene capacidad para dos unidades de disco

DOCUMENTACION
El manual que se proporciona es muy irregular; en ocasiones resulta difícil localizar la información que se desea

VENTAJAS
La provisión de CP/M y del Dr LOGO coloca al ordenador Amstrad al nivel de una máquina muy potente

DESVENTAJAS
El AmsDOS no está muy bien implementado y los usuarios que dispongan de una sola unidad encontrarán que ni el AmsDOS ni el CP/M son totalmente adecuados para sus necesidades

El LOGO permite emplear las capacidades de sonido y gráficos del Amstrad y la implementación les resultará familiar a quienes hayan seguido nuestra serie dedicada al LOGO. También se ha previsto que el lenguaje sea operado mediante una palanca de mando y pulsadores de disparo, de modo que es posible escribir juegos en LOGO que se puedan desarrollar fácilmente mediante control por palanca de mando.

Si bien la implementación del Dr LOGO es muy buena y sale bien parada en comparación con otras muchas versiones del lenguaje, existen ciertas dudas sobre cómo se desempeñará bajo las limitaciones del hardware. Por ejemplo, un programa de teselado (uno que exige que el mismo diseño se repita varias veces con el fin de llenar la pantalla) que funciona a la perfección en el Commodore 64, generó en el Amstrad un mensaje "no hay suficiente espacio en la pila", lo que indica que quizá el lenguaje no quepa tan cómodamente como debiera. Aparte de esto, el Dr LOGO es rápido y amable y es una introducción ideal al lenguaje.

Tal como sucede con todos los sistemas de disco para micros personales, la popularidad del sistema, al menos al principio, dependerá del apoyo de software. Amstrad ya ha realizado numerosas utilidades para la unidad de disco, la mayoría de las cuales parecen dirigidas al aficionado personal más "serio". Estos paquetes incluyen una base de datos, un procesador de textos y lenguajes adicionales como PASCAL.

Al lanzar la unidad de disco para su máquina, Amstrad ha dado un paso evidentemente exitoso para hacer de su ordenador una máquina más acorde con el mercado sin imponer el precio que normalmente está asociado a tales sistemas. La adición de una unidad de disco ortoga al ordenador aún más competitividad y refleja las intenciones de Amstrad de ensanchar el horizonte de la máquina más allá del desfalleciente mercado de juegos.

Asegurando las escotillas

Prosiguiendo con nuestro proyecto de programación, nos ocuparemos del código de los cuatro restantes eventos mayores que se desarrollan antes de llegar a destino

El programa es dirigido al azar hacia una subrutina de contingencia mayor mediante la subrutina de la línea 6500, que genera un número aleatorio y lo utiliza en la sentencia ON X GOSUB de la línea 6510, descrita en el módulo anterior. Para dar cabida a la cantidad de posibilidades de este módulo, es necesario añadir a esta línea los primeros números de línea de las cuatro contingencias extras. La línea 6510 debe rezar ahora:

```
6510 ON X GOSUB 6530,6700,6800,6900,7000,
7050
```

El tercer número de línea llama a una subrutina en la cual el barco es atacado por piratas. El programa comprueba si esta contingencia ya se ha producido examinando el valor del indicador de eventos para esta subrutina, M(3). Si M(3) es 1, retorna al programa principal; de lo contrario continúa, poniéndolo a 1 en la línea 6818 para evitar la repetición.

Es posible que todos los tripulantes estén muertos, razón que haría irrelevante el ataque de los piratas. El programa prepara un bucle para comprobar los valores de la matriz de fortaleza de la tripulación, y cuenta los tripulantes con fortalezas de 0 o −999 examinando la tasa de fortaleza, TS(T,2), la matriz de fortaleza/categoría. Los tripulantes fallecidos o inexistentes se cuentan en X, y si X es igual a 16 no queda ninguno y el control se devuelve al programa principal.

Si la contingencia no se ha implementado con anterioridad y si quedan tripulantes a bordo, los piratas atacan, matando a algunos de ellos. La cantidad de muertos dependerá de que la tripulación disponga de armas para defenderse. La cuenta de los muertos se registrará en K, siendo dos el número mínimo. La línea 6825 establece el valor de K en esta cifra, y el programa examina entonces el elemento apropiado de la matriz de provisiones, OA(2), para ver si hay algún arma a bordo. Si el valor de OA(2) es 0 o −999, no hay ningún arma y K se establece en 4. Si hay armas a bordo, la variable S$ se establece en A PESAR DE TUS ARMAS. Si K se hubiera establecido en 4, S$ cambia por NO TIENES ARMAS.

Entre las líneas 6836 y 6845 se prepara otro bucle para suprimir el número adecuado de tripulantes, y la línea 6835 iguala X a 0 para conservar una cuenta de esta cifra. Si el valor de fortaleza de un determinado elemento es 0 o −999, el bucle pasa a considerar el siguiente elemento; si no lo es, la línea 6840 establece el valor en −999 e incrementa el valor de X en 1. Cuando el valor de X resulta igual al número de tripulantes que serán muertos, K, la línea 6842 establece el contador del bucle, T, en 16 y el programa sale del bucle. Si no quedan K tripulantes que matar, el programa saldrá del bucle tras efectuar la búsqueda 16 veces. Se imprime el número de tripulantes muertos, para completar la frase iniciada ya sea en la línea 6828 o bien en la 6830. Se solicita entonces al jugador que pulse cualquier tecla para continuar.

Rotura del timón

El siguiente acontecimiento consiste en la rotura del timón, que hace necesaria una reparación. Si se ha contratado algún mecánico, y éste está aún con vida, el timón se reparará rápidamente y el viaje se reanudará tras una breve demora; pero si no hay ningún mecánico a bordo, la travesía, como es lógico, durará más. La subrutina de rotura del timón comienza en la línea 6900 y es el cuarto número de línea de la sentencia ON X GOSUB de la línea 6510. Nuevamente, la subrutina comprueba si este evento ya se ha producido mediante el método usual: ver si el valor de M(4) se ha establecido en 1 y, si no fuera así, establecerlo en 1.

El valor de X se establece en 4, indicando la cantidad de semanas extras que durará el viaje si no hay ningún mecánico disponible. El programa revisa entonces la matriz de fortaleza/categoría, TS(,), en busca de un mecánico vivo. Entre las líneas 6930 y 6938 se establece un bucle que busca el número 3 en la matriz de categoría de tripulación de TS(,), que representa un mecánico, y un valor para la fortaleza del mecánico que no sea igual a 0 ni a −999, lo que significaría que el mecánico estaría vivo. Si se satisfacen todas estas condiciones, entonces el valor de X se restablece de 4 a 1. De haber disponible un mecánico la avería del timón se reparará enseguida y la duración del viaje se incrementará en sólo una semana.

No obstante, se le dice al jugador: AUNQUE TIENES UN MECANICO o, si X es igual a 4, NO TIENES NINGUN MECANICO Y. Entonces se imprime la duración extra de la travesía. Este valor se suma a la duración total del viaje, EW, en la línea 6965 y en este momento el juego continuará pulsando cualquier tecla. La variable X tiene, por consiguiente, dos finalidades: primero, indica el número de semanas a sumar al viaje, pero también actúa a modo de bandera, indicando si hay o no un mecánico disponible.

Una tormenta, manipulada por la subrutina de la línea 7000, constituye el quinto evento mayor posible. Desvía al barco de su curso y aumenta la duración del viaje. Si entre la tripulación hay algún oficial, el barco recuperará pronto el rumbo correcto.

Ian McKinnell

De no ser así, el viaje durará más tiempo. La estructura de esta subrutina es similar a la subrutina del "timón roto" y, de hecho, utiliza parte de su código.

El programa comprueba si la tormenta ya se ha producido y, en caso negativo, establece a 1 el valor de M(5). La duración extra del viaje se registra en el valor de X, que se establece en 2, y si no hay ningún oficial a bordo la travesía dura dos semanas más. Entre las líneas 7030 y 7038 se prepara un bucle para explorar la matriz de la tripulación en busca de un oficial, buscando un valor de 4 en la matriz de categoría, y un valor mayor que 0 en la matriz de fortaleza. Si se satisfacen estas exigencias, el valor de X se establece en 1, y T se establece en 16, haciendo que el programa salga del bucle.

S$ se establece en A PESAR DE QUE CUENTAS CON UN OFICIAL o en NO TIENES NINGUN OFICIAL Y, según el valor de X. El programa pasa entonces a la línea 6950, de la subrutina de rotura del timón, para utilizar de nuevo la sección del código que imprime la duración extra de la travesía, incrementa EW, la duración del viaje, en X, el tiempo extra, y retorna al programa principal.

¡Tierra, tierra!

El último de los acontecimientos mayores, avistar una isla, se produce en la subrutina que empieza en la lína 7050. Éste es el último número de línea de la sentencia ON X GOSUB de la línea 6510. La isla no se halla en la ruta principal y, por tanto, una visita a la misma prolongaría la duración del viaje. No obstante, cambiando el rumbo y desembarcando, el barco podría reabastecerse de algunas provisiones. Por consiguiente, el jugador debe tomar la decisión de poner rumbo a tierra o continuar su recorrido actual. Si el barco visita la isla, el resultado de la expedición en busca de agua y comida podría verse fuertemente influido por cualquier encuentro previo con el albatros. ¡Si el jugador hubiera abatido al albatros el desvío no tendría ningún éxito!

El programa comprueba si la subrutina ya se ha ejecutado previamente, de la forma habitual. Se le informa entonces al jugador que las cartas indican la existencia de una isla en donde se podría reabastecer de provisiones y que el desvío prolongaría la duración del viaje, y se le pregunta si va a realizar la visita o no. La línea 7082 aguarda una respuesta y analiza el primer carácter de la entrada para determinar si la respuesta es sí o no. Si el jugador no desea alejarse de su rumbo, el control retorna al programa principal mediante la línea 7086. Si la respuesta es SI o S, el jugador arriba a la isla en la línea 7100.

El programa comprueba entonces si el jugador ha abatido al albatros, en la línea 7106. Si el albatros no fue muerto, B$, que se estableció en N en la línea 48, no se modificará y el programa pasará a la línea 7110. Sin embargo, si el jugador abatió al ave, B$ se habrá establecido en S mediante la línea 6162 de la subrutina del albatros. Si el ave fue muerta, la isla será una tierra yerma y el agua estará envenenada, no se conseguirán provisiones y se le recordará al jugador la circunstancia del disparo. El programa es enviado hasta la línea 7130, que incrementa la duración del viaje.

Si el albatros no fue derribado, en la línea 7110 se prepara un bucle de 1 a 4 para cada tipo de provisión, que recogerá las provisiones extras; la cantidad de suministros se selecciona al azar. La línea 7112 genera un número aleatorio y, si éste es menor que .25, pasa a la provisión siguiente. (Existe una posibilidad entre cuatro de no conseguir cada uno de los tipos de provisiones.) La línea 7115 genera un número aleatorio entre 5 y 14, almacenado en X. La línea 7120 imprime esta cantidad, seguida por las unidades de esa provisión en particular, ya sea en kilos o en barriles. Si durante la semana actual alguna provisión hubiera sido arrastrada por la borda, en la matriz de provisiones la cantidad se habría establecido en −999. De ser así, la línea 7122 restablece el valor a 0 para permitir la adición de las provisiones extras. Las provisiones extras, X, se suman a las provisiones existentes mediante la línea 7125. La travesía se incrementará en una o dos semanas. La línea 7135 decide de forma aleatoria cuál será el incremento, se le informa al jugador y se añade el tiempo extra a la variable de duración del viaje, EW, en la línea 7140.

Módulo 9: Otros eventos mayores

Inicializar banderas

```
48 A$="N":B$="N"
```

Rutina Contingencia de piratas

```
6800 REM PIRATAS
6805 IF M(3)=1 THEN RETURN
6810 X=0
6812 FOR T=1 TO 16
6814 IF TS(T,2)=0 OR TS(T,2)=-999 THEN X=X+1
6815 NEXT
6816 IF X=16 THEN RETURN
6818 M(3)=1
6820 PRINT CHR$(147)
6822 S$=" EL BARCO ES ATACADO POR PIRATAS!*":GOSUB 9100
6824 PRINT:GOSUB 9200
6825 K=2
6826 IF OA(2)=0 OR OA(2)=-999 THEN K=4
6828 S$="A PESAR DE TUS ARMAS*"
6830 IF K=4 THEN S$="NO TIENES ARMAS *"
6832 GOSUB 9100
6835 X=0
6836 FOR T=1 TO 16
6838 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 6845
6840 X=X+1:TS(T,2)=-999
6842 IF X=K THEN T=16
6845 NEXT
6850 PRINT X;
6855 S$="TRIPULANTE HA RESULTADO MUERTO*"
6856 IF X>1 THEN S$="TRIPULANTES HAN RESULTADO MUERTOS*"
6860 GOSUB 9100
6865 PRINT:GOSUB 9200
6890 S$=K$:GOSUB 9100
6895 GET I$:IF I$="" THEN 6895
6899 RETURN
```

Rutina Contingencia del timón

```
6900 REM TIMON
6905 IF M(4)=1 THEN RETURN
6910 PRINT CHR$(147)
6915 M(4)=1
6920 S$="HAY PROBLEMAS CON EL TIMON!*":GOSUB 9100
6925 PRINT:GOSUB 9200
6928 X=4
6930 FOR T=1 TO 16
6935 IF TS(T,1)=3 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN
     X=1:T=16
6938 NEXT
6940 S$="A PESAR DE QUE TIENES UN MECANICO*"
6945 IF X=4 THEN S$="NO TIENES MECANICO Y*"
6950 GOSUB 9100
6955 S$="TU VIAJE DURARA*":GOSUB 9100
6960 PRINT X;"SEMANAS MAS"
6965 EW=EW+X
6967 PRINT:GOSUB 9200
6969 S$=K$:GOSUB 9100
6970 GET I$:IF I$="" THEN 6970
6975 RETURN
```

Rutina Contingencia de la tormenta

```
7000 REM TORMENTA
7005 IF M(5)=1 THEN RETURN
7010 PRINTCHR$(147)
7015 M(5)=1
7020 S$="EL VIENTO TE APARTA DE TU RUMBO*":GOSUB 9100
7022 S$="DURANTE UNA TORMENTA!*":GOSUB 9100
7025 PRINT:GOSUB 9200
7028 X=2
7030 FOR T=1 TO 16
7035 IF TS(T,1)=4 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN
     X=1:T=16
7038 NEXT
7040 S$="A PESAR DE QUE CUENTAS CON UN OFICIAL*"
7045 IF X=2 THEN S$="NO TIENES NINGUN OFICIAL Y*"
7049 GOTO 6950
```

Rutina Contingencia de la isla

```
7050 REM ISLA
7055 IF M(6)=1 THEN RETURN
7060 PRINTCHR$(147)
7065 M(6)=1
7070 S$="TUS CARTAS INDICAN LA EXISTENCIA DE UNA ISLA*":GOSUB
     9100
7071 S$="EN LA QUE PODRIAS*":GOSUB 9100
7072 S$="REABASTECERTE DE PROVISIONES*":GOSUB 9100
7073 S$="PERO SI TE DESVIAS HACIA ELLA*":GOSUB 9100
7074 S$="SUPONDRA UNA MAYOR DURACION DEL VIAJE*":GOSUB 9100
7075 PRINT:GOSUB 9200
7080 S$="QUIERES IR A LA ISLA?*":GOSUB 9100
7082 INPUT I$:I$=LEFT$(I$,1)
7084 IF I$<>"S" AND I$<>"N" THEN 7082
7086 PRINT:GOSUB 9200
7090 IF I$="N" THEN 7145
7100 S$="LLEGAS A LA ISLA*":GOSUB 9100
7105 S$="Y CONSIGUES*":GOSUB 9100
7106 IF B$="N" THEN 7110
7107 PRINT:GOSUB 9200
7108 PRINT"NADA!":GOSUB 9200
7109 S$="(RECUERDA EL ALBATROS!)":GOSUB 9100;GOTO 7130
7110 FOR T=1 TO 4
7112 IF RND(1)<.25 THEN 7129
7115 X=INT(RND(1)*10)+5
7120 PRINT X;U$(T);"S DE";P$(T)
7122 IF PA(T)=-999 THEN PA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="PERO AHORA EL VIAJE DURARA*":GOSUB 9100
7135 X=INT(RND(1)*2)+1
7139 PRINT X;:S$="SEMANAS MAS*":GOSUB 9100
7140 EW=EW+X
7145 PRINT:GOSUB 9200
7150 S$=K$:GOSUB 9100
7155 GET I$:IF I$="" THEN 7155
7159 RETURN
```

Complementos al BASIC

Spectrum:
Introducir las siguientes modificaciones:

```
6512 IF X=3 THEN GOSUB 6800
6513 IF X=4 THEN GOSUB 6900
6514 IF X=5 THEN GOSUB 7000
6515 IF X=6 THEN GOSUB 7050

6820 CLS
6895 LET I$=INKEY$:IF I$=""THEN GO TO 6895
6910 CLS
6970 LET I$=INKEY$:IF I$=""THEN GO TO 6970
7010 CLS
7060 CLS
7082 INPUT I$:LET I$=I$(1 TO 1)
7155 LET I$=INKEY$:IF I$=""THEN GO TO 7155
```

BBC Micro:
Introducir las siguientes modificaciones:

```
6820 CLS
6895 I$=GET$
6910 CLS
6970 I$=GET$
7010 CLS
7060 CLS
7155 I$=GET$
```

Gran Premio 2

He aquí la segunda parte de un juego del cual ya hemos proporcionado un programa para su ordenador Atari. No olvide que la palanca de mando es obligatoria

Para conducir su bólido utilice la palanca de mando. Si logra llegar a la meta, será bonificado: el valor de la distancia recorrida se duplicará.

```
10 REM ** GRAN PREMIO 2 - P.BUNN **
15 PI=5:GOSUB 3000:ACCIDENTE=1000
20 X2=14:X=116:S=PEEK(106)-8:Q=S*256:FOR
   N=Q+512 TO Q+640:POKE N,0:NEXT N:POKE
   54279,S
30 POKE 559,46:POKE 53248,X:POKE 704,216:POKE
   704,70:POKE 53256,1
40 FOR N=Q+552 TO Q+561:READ A:POKE N,A:NEXT N
50 DATA 129,195,165,24,24,153,219,165,36,24
60 GRAPHICS 0:SETCOLOR 2,0,0:POKE 53277,3:POKE
   559,46
65 POKE 752,1:POKE 53278,A
66 FOR P=0 TO 23:POKE 201,X2:? ,S2$:NEXT P
70 S=STICK(0):X=X+(S=7)*2:X=X-(S=11)*2:POKE
   53248,X
72 SOUND 0,40,10,15:SC=SC+PI:SOUND 0,0,0,0
75 IF O THEN O=0:Z=2:GOTO 110
80 A=PEEK(53770)
90 IF A>85 AND A<170 THEN Z=2
100 IF A>170 AND X2<15 THEN Z=3:O=1
105 IF A<85 AND X2>2 THEN X2=X2-1:Z=1:O=1
110 POKE 201,X2
115 IF A=192 AND NOT I THEN ? ,F$:I=1:GOTO 145
120 IF Z=1 THEN ? ,S1$
130 IF Z=2 THEN ? ,S2$
140 IF Z=3 THEN ? ,S3$:X2=X2+1
145 Y=PEEK(53252):IF Y<>0 AND I THEN GOTO 2000
150 IF Y<>0 THEN GOTO ACCIDENTE
160 GOTO 70
1000 REM ** ACCIDENTE!!!! **
1010 N=INT(RND(0)*10):POKE
     Q+552+N,PEEK(53770):SOUND
     0,RND(0)*20+20,80,15:POKE 704,PEEK(53770)
1020 IF PEEK(53770)<240 THEN 1010
1030 POKE 53248,0:? :? "SUS PUNTOS:",SC :? :? :? "PULSE
     UNA TECLA.":POKE 764,255
1035 SOUND 0,0,0,0
1040 IF PEEK(764)=255 THEN 1040
1050 RUN
2000 FOR Y=0 TO 255:POKE 710,Y:NEXT Y
2005 ? CHR$(125);"       PUNTOS:";SC:POKE 710,0
2010 ?"       BONO   :1000":B=1000
2020 FOR G=1 TO 1000 STEP
     10:B=B-10:SC=SC+10:POSITION 14,0:? SC:POSITION
     14,1:? B;"    ":SOUND 0,G/4,10,10:NEXT G
2030 SOUND 0,0,0,0
2040 ? :PI=PI*2:? "1 KM VALE    ";PI;" PUNTOS"
2045 RESTORE :I=0
2050 GOTO 20
3000 DIM F$(8),S1$(8),S2$(8),S3$(8)
3010 RESTORE 3000:FOR P=1 TO 8
3020 READ A,B,C,D
3030 F$(P,P)=CHR$(A)
3040 S1$(P,P)=CHR$(B)
3050 S2$(P,P)=CHR$(C)
3060 S3$(P,P)=CHR$(D)
3070 NEXT P
3080 RESTORE :RETURN
3090 DATA 160,6,22,7,198,32,32,32
3100 DATA 201,32,32,32,206,32,32,32
3110 DATA 201,32,32,32,211,32,32,32
3120 DATA 200,32,2,32,160,6,32,7
```

Para el Spectrum

Ahora ofrecemos para el Spectrum un programa que permite controlar el brazo-robot a través de la interface construida anteriormente

El software requerido para controlar el brazo-robot debe basarse en código máquina, porque los servomotores digitales empleados requieren impulsos regulares cada quincuagésima de segundo. Tales velocidades se hallan fuera del alcance de un programa en BASIC. Al igual que las versiones para el Commodore y el BBC Micro, este código máquina se ejecuta sobre una base de interrupciones, pero difiere significativamente porque el Spectrum utiliza un procesador Z80 en vez de los de la serie 65XX que emplean las otras dos máquinas. La mejor forma de tratar interrupciones en código máquina de Z80 es utilizar interrupciones en modo IM 2. Para explicarlo en términos simples, cuando se establece esta modalidad de interrupciones y se genera una interrupción para actualizar la visualización en pantalla (afortunadamente, alrededor de cada quincuagésima de segundo), el procesador toma la dirección de inicio del código de servicio de interrupción de dos fuentes. El byte *hi* de la dirección está retenido en el registro I y el byte *lo* se toma del estado del bus de datos cuando se produjo la interrupción.

El sistema está diseñado para comunicaciones con periféricos conectados al bus de datos que puedan controlar el valor retenido en el bus. Sin embargo, en nuestra aplicación no conocemos el estado del bus de datos y, por lo tanto, hemos de acomodar las 256 posibilidades. El programa, por consiguiente, prepara una página de memoria para retener la dirección de comienzo real de nuestro programa. Llenar una página con #FBs y apuntar a la página mediante el establecimiento del registro I provoca una interrupción IM 2 que empezará a ejecutar código en la posición #FBFB. (Esta técnica se describirá en profundidad en una futura serie de código máquina dedicada al OS del Spectrum.)

El resto del código máquina utiliza los mismos principios de operación que en las versiones 65XX, produciendo una corriente de impulsos hacia la puerta E/S 31, la puerta asociada a nuestra interface. El programa de control en BASIC, asimismo, sigue líneas similares a las desarrolladas para el BBC Micro y el Commodore 64, permitiendo la programación de secuencias desde el teclado y su ulterior reproducción.

Programador de secuencias para el brazo en el Spectrum

Cargador en BASIC

```
 10 REM ************************************************
 15 REM ************************************************
 17 REM **                                            **
 20 REM **                  controlador               **
 25 REM **                  brazo spectrum            **
 30 REM **                                            **
 40 REM ************************************************
 50 REM ************************************************
 60 :
 70 CLEAR 30000
 80 LET sa=64017:REM comienzo c/m cuña
 82 LET np=64008:REM direccion comienzo nuevapos
 84 LET dl=64016:REM direccion factor demora
 86 LET en=64148:REM cuña apagada
 88 LET sm=64151:REM dir comienzo movimiento uniforme
 90 LOAD "ARM.HEX"CODE sa
100 GO SUB 1000:REM preparacion
110 CLS
120 PRINT AT 6,2;"te gustaria:"
130 PRINT AT 9,2;"1..programar nueva secuencia"
140 PRINT AT 10,2;"2..añadir movimientos a un programa"
150 PRINT AT 11,2;"3..cargar un archivo"
160 PRINT AT 12,2;"4..salir del programa"
170 LET g$=INKEY$:IF g$="" THEN GO TO 170
180 IF g$="1" THEN LET c=1:LET lm=0
190 IF g$="1" OR g$="2" THEN GO SUB 2000:GO SUB 3000:GO SUB
    4000:GO SUB 5000
200 IF g$="3" THEN GO SUB 6000
210 IF g$="4" THEN CLS:RANDOMIZE USR en:STOP
220 GO TO 110
230 :
1000 REM **** preparacion ****
1010 LET c=1:LET ns=4:REM n. de servos
1020 LET lm=0:LET oc=0:LET df=10:REM factor de demora
1030 LET mc=100:REM max. contador
1040 DIM r(mc,ns):REM matriz posiciones teclas
1050 DIM l(1):REM matriz datos ultimomax
1060 FOR i=0 TO 3:POKE np+i,0:NEXT i
1070 RANDOMIZE USR sa
1090 RETURN
1099 :
2000 REM **** informe ****
2010 CLS
2020 PRINT AT 3,2;"por favor utiliza"
2030 PRINT AT 4,2;"n/m...para i/d"
2040 PRINT AT 5,2;"p/l...para 1.er brazo ar/ab"
2050 PRINT AT 6,2;"a/z...para 2.º brazo ar/ab"
2060 PRINT AT 7,2;"x/c...para pinza abrir/cerrar"
2070 PRINT AT 8,2;"s.....guardar una posicion"
2080 PRINT AT 9,2;"q.....retornar al menu"
2090 PRINT AT 10,2;"r.....mover a posicion guardada"
2100 PRINT AT 11,2;"v/b...siguiente y anterior contador"
2110 PRINT AT 12,2;"e.....establecer nuevo contador"
2120 PRINT AT 13,2;"i/d...para aum/dism velocidad"
2130 PRINT AT 1,2;"contador=";c;"  ":
2135 FOR i=1 TO 4:PRINT r(c,i);"  ";:NEXT i:PRINT
2140 RETURN
2190 :
3000 REM **** programar brazo ****
3010 POKE dl,1
3020 LET dx=5
3030 LET a$=INKEY$: IF a$="" THEN GO TO 3030
3035 IF a$="n" THEN LET p=PEEK(np)+dx:IF p<256 THEN POKE np,p
3040 IF a$="m" THEN LET p=PEEK (np)-dx:IF p>0 THEN POKE np,p
3050 IF a$="p" THEN LET p=PEEK (np+1)+dx: IF p<256 THEN POKE
     np+1,p
3060 IF a$="l" THEN LET p=PEEK(np+1)-dx:IF p>0 THEN POKE np+1,p
3065 IF a$="a" THEN LET p=PEEK (np+2)+dx:IF p<256 THEN POKE
     np+2,p
3070 IF a$="z" THEN LET p=PEEK (np+2)-dx:IF p>0 THEN POKE
     np+2,p
3080 IF a$="x" THEN LET p=PEEK (np+3)+3*dx:IF p<256 THEN POKE
     np+3,p
3090 IF a$="c" THEN LET p=PEEK (np+3)-3*dx:IF p>0 THEN POKE
     np+3,p
3100 IF a$="r" THEN FOR i=1 TO 4:POKE np+i-1,r(c,i):NEXT i
3110 IF a$="v" THEN LET c=c+1:IF c>mc THEN LET c=1
3120 IF a$="b" THEN LET c=c-1:IF c<1 THEN LET c=mc
3130 IF a$="e" THEN PRINT AT 2,20;"valor contador":INPUT c
3140 IF A$="s" THEN FOR i=1 TO 4:LET r(c,i)=PEEK (np+i-1):NEXT i:
     LET c=c+1
3145 IF a$="i" THEN LET dx=dx+1
3147 IF a$="d" THEN LET dx=dx-1:IF dx<1 THEN LET dx=1
3150 IF c>lm THEN LET lm=c
3160 IF oc<>c AND c>1 THEN PRINT AT 1,2;"contador=";c-1;"  ";:FOR
     i=1 TO 4:PRINT r(c-1,i);"  ";: NEXT i:PRINT
3170 LET oc=c
3190 IF a$<>"q" THEN GO TO 3030
3200 RETURN
3900 :
4000 REM **** reproducir secuencia ****
4010 CLS
4020 PRINT AT 12,2;"reproducir secuencia s/n, r repite"
4030 LET a$="INKEY$:IF a$<>"s" AND a$<>"n" AND a$<>"r" THEN
     GO TO 4030
4040 IF a$="n" THEN RETURN
4050 IF a$="r" THEN PRINT AT 14,1;"factor de demora 1-255";:INPUT df
4060 IF df<1 OR df>255 THEN GO TO 4050
4070 POKE dl,df:REM establecer registro demora
4080 FOR i=l TO lm
4090 PRINT AT 1,2;"n.en secuencia=";i;"  ";
4100 FOR s=1 TO 4
4110 POKE np+s-1,r(i,s)
```

```
4120 NEXT s:NEXT i
4140 GO TO 4010
4990 :
5000 REM **** guardar un archivo ****
5010 CLS
5020 PRINT AT 12,2;"guardar secuencia (s/n)?"
5030 LET g$=INKEY$:IF g$<>"s" AND g$<>"n" THEN GO TO 5030
5040 IF g$="n" THEN RETURN
5050 LET l(1)=lm
5060 INPUT "nombre archivo";f$
5065 PRINT "pulsar play y record"
5067 RANDOMIZE USR en:REM cuña apagada
5070 SAVE f$+".lm" DATA l()
5080 SAVE f$+".r" DATA r()
5085 RANDOMIZE USR sa:REM cuña encendida otra vez
5090 RETURN
5900 :
6000 REM **** cargar un archivo ****
6010 CLS
6020 PRINT AT 12,2;"cargar un archivo (s/n)?"
6030 LET g$=INKEY$:IF g$<>"s" AND g$<>"n" THEN GO TO 6030
6040 IF g$="n"THEN RETURN
6045 INPUT "nombre archivo";f$
6050 FOR i=1 TO mc
6060 FOR j=1 TO ns
6070 LET r(i,j)=0
6080 NEXT j:NEXT i
6090 RANDOMIZE USR en:REM cuña apagada
6100 LOAD f$+".lm" DATA l()
6110 LET lm=l(1):LET c=lm:LET oc=0
6120 LOAD f$+".r" DATA r()
6125 RANDOMIZE USR sa:REM cuña encendida otra vez
6130 RETURN
```

Listado assembly

```
                    1000 ;CONTROLADOR BRAZO SPECTRUM
                    1010
001F                1020 PORT:      EQU     31
F900                1030            ORG     # F900
F900                1040 MOTTAB     DEFS    256
FA00                1050 ANGTAB     DEFS    8
FA08                1060 NEWPOS     DEFS    8
FA10                1070 DELAY:     DEFS    1
                    1080
                    1090 ;PREPARAR TABLA VECTOR
FA11  2100FC        1100 INIT:      LD      HL,#FC00
FA14  01FB00        1110            LD      BC,#00FB
FA17  71            1120 LOOP1:     LD      (HL),C
FA18  23            1130            INC     HL
FA19  10FC          1140            DJNZ    LOOP1
FA1B  71            1150            LD      (HL),C
FA1C  3EFC          1170            LD      A,#FC
FA1E  ED47          1180            LD      I,A
                    1190 ;INIT TABLAS A FF
                    1200 ;
FA20  2100F9        1210            LD      HL,MOTTAB
FA23  3EFF          1220            LD      A,#FF
FA25  06FF          1230            LD      B,#FF
FA27  77            1240 LOOP2:     LD      (HL),A
FA28  2C            1250            INC     L
FA29  10FC          1260            DJNZ    LOOP2
FA2B  2100FA        1270            LD      HL,ANGTAB
FA2E  0610          1280            LD      B,16
FA30  77            1290 LOOP2A     LD      (HL),A
FA31  2C            1300            INC     L
FA32  10FC          1305            DJNZ    LOOP2A
FA34  ED5E          1310            IM      2
FA36  C9            1320            RET
                    1380 ; MANEJADOR INTERRUPCIONES
                    1390
FA37  F3            1410 HANDLE     DI
FA38  F5            1420            PUSH    AF
FA39  C5            1430            PUSH    BC
FA3A  D5            1440            PUSH    DE
FA3B  E5            1450            PUSH    HL
FA3C  FF            1460            RST     #38;RUTINA NML
FA3D  F3            1470            DI
FA3E  CD47FA        1480            CALL    EVENT ;NUESTRA RUTINA
FA41  E1            1490            POP     HL
FA42  D1            1500            POP     DE
FA43  C1            1510            POP     BC
FA44  F1            1520            POP     AF
FA45  FB            1530            EI
FA46  C9            1540            RET
                    1550
                    1560 ;
                    1570 ;++++ RUTINA EVENT ++++
                    1580 ;
FA47  0608          1590 EVENT:     LD      B,8
FA49  3E7F          1600            LD      A,#7F
FA4B  16F0          1610            LD      D,#F0
FA4D  2158FA        1620            LD      HL,FIX+2
FA50  DD2100FA      1630            LD      IX,ANGTAB
FA54  3607          1635            LD      (HL),7
FA56  DD5E07        1640 FIX:       LD      E,(IX+7)
FA59  12            1650            LD      (DE),A
FA5A  0F            1660            RRCA
FA5B  35            1670            DEC     (HL)
FA5C  10F8          1680            DJNZ    FIX
FA5E  0600          1690 ;
FA60  3EFF          1740 ;PREP MOTTAB PARA PORTSEND
FA62  2100F9        1750 ;
FA65  A6            1760            LD      B,0
FA66  77            1770            LD      A,#FF
FA67  2C            1780            LD      HL,MOTTAB
FA68  10FB          1790 LOOP3:     AND     (HL)
                    1800            LD      (HL),A
                    1810            INC     L
                    1820            DJNZ    LOOP3
                    1830 ;
                    1840 ;++++ COMENZAR IMPULSOS ++++
                    1850 ;
FA6A  3EFF          1860            LD      A,#FF
FA6C  D31F          1870            OUT     (PORT),A
                    1880 ;
                    1885 ;++++ LLAMAR DEMORA ++++
FA6E  D5            1887            PUSH    DE
FA6F  1E04          1889            LD      E,4
FA71  16FF          1890            LD      D,#FF
FA73  CD8BFA        1892            CALL    OLOOP
FA76  D1            1894            POP     DE
                    1920 ;
                    1930 ;++++ ENVIAR MOTTAB A PORT ++++
                    1940 ;
FA77  0E1F          1950            LD      C,PORT
FA79  06FF          1960            LD      B,#FF
FA7B  2E00          1970            LD      L,00
FA7D  EDB3          1980            OTIR
                    1990 ;
                    2000 ;++++ RESTAURAR MOTTAB A FF ++++
+
FA7F  3EFF          2010            LD      A,#FF
FA81  0600          2020            LD      B,0
FA83  2100F9        2030            LD      HL,MOTTAB
FA86  77            2040 LOOP4:     LD      (HL),A
FA87  2C            2050            INC     L
FA88  10FC          2060            DJNZ    LOOP4
FA8A  C9            2070            RET
                    2080 ;++++ BUCLE DEMORA ++++
                    2090 ;
FA8B  1D            2100 OLOOP:     DEC     E
FA8C  C8            2110            RET     Z
FA8D  15            2120 ILOOP:     DEC     D
FA8E  CA8BFA        2130            JP      Z, OLOOP
FA91  C38DFA        2140            JP      ILOOP
                    2200 ;
                    2210 ;++++ RESTAURAR IM 1 ++++
                    2220
FA94  ED56          2230 REST:      IM      1
FA96  C9            2240            RET
                    2250 ;
                    2340 ;++++ MOVEDOR UNIFORME ++++
                    2350 ;
FA97  0E04          2360 START:     LD      C,4
FA99  0604          2370            LD      B,4     ;PREPARAR
                                                    CONTADORES
FA9B  2130FA        2380            LD      HL,ANGTAB+3
FA9E  110BFA        2390            LD      DE,NEWPOS+3
FAA1  7E            2400 NEXMOT     LD      A,(HL)
FAA2  EB            2410            EX      DE,HL
FAA3  BE            2420            CP      (HL)
FAA4  EB            2430            EX      DE,HL
FAA5  280A          2440            JR      Z,MOVED
FAA7  3804          2450            JR      C,ADD
FAA9  35            2460            DEC     (HL)
FAAA  C3B2FA        2470            JP      NEXT
FAAD  34            2480 ADD:       INC     (HL)
FAAE  C3B2FA        2490            JP      NEXT
FAB1  0D            2500 MOVED:     DEC     C
FAB2  2B            2510 NEXT:      DEC     HL
FAB3  1B            2520            DEC     DE
FAB4  10EB          2530            DJNZ    NEXMOT
FAB6  F5            2540            PUSH    AF
FAB7  D5            2550            PUSH    DE
FAB8  ED5B10FA      2560            LD      DE,(DELAY)
FABC  16FF          2570            LD      D,#FF
FABE  CD8BFA        2580            CALL    OLOOP ;LLAMAR DEMORA
FAC1  D1            2590            POP     DE
FAC2  F1            2600            POP     AF
FAC3  20D2          2610            JR      NZ,START
FAC5  C9            2620            RET
                    3000 ;++++ PREPARAR MANEJADOR ++++
                    3010 ;++++ DIRECCION DE SALTO ++++
                    3020 ;
FBFB                3030            ORG     #FBFB
FBFB  F3            3040            DI
FBFC  C337FA        3050            JP      HANDLE
Pass 2 errors:      00

ADD     FAAD    ANGTAB  FA00
DELAY   FA10    EVENT   FA47
FIX     FA56    HANDLE  FA37
ILOOP   FA8D    INIT    FA11
LOOP1   FA17    LOOP2   FA27
LOOP2A  FA30    LOOP3   FA65
LOOP4   FA86    MOTTAB  F900
MOVED   FAB1    NEWPOS  FA08
NEXMOT  FAA1    NEXT    FAB2
OLOOP   FA8B    PORT    001F
REST    FA94    START   FA97
```

Imágenes a trozos

Vamos a analizar la técnica de producir al mismo tiempo dibujo y texto, tan frecuente en programas de juegos

Para visualizar los datos de video, el chip VIC-II debe leerlos de la memoria. Esto se consigue haciendo que el VIC-II acceda a algunas líneas (no todas) del bus de direcciones, y a todas las líneas del bus de datos. El proceso por el cual el VIC-II lee la ROM y la RAM compartida con el 6510 se denomina *acceso directo a memoria* (DMA). El VIC-II debe, en teoría, usar las líneas de dirección o de datos en los ratos en que el 6510 no las utilice, en cuyo caso la actuación de ambos procesadores es "transparente" recíprocamente. Pero el 6510 es un procesador bastante modesto con muy pocos registros internos y ninguna de las instrucciones del 6510 consume más de siete ciclos de reloj (algunas tan sólo tres o cuatro).

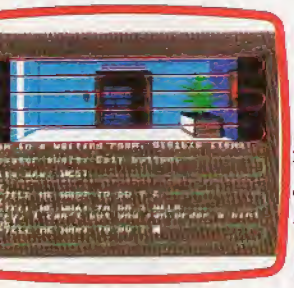
Ya que el VIC-II debe leer todo un conjunto de datos, sobre todo cuando visualiza varios sprites, no hay tiempo material para que pueda actuar de la forma transparente antedicha. Por ello, el VIC-II tiene una línea de control especial, llamada de *bus disponible* (BA: *bus available*) que puede emplear cuando desee enviar una señal de "precedencia" al 6510. Lo cual le permite reservarse tanto tiempo como necesite para leer los datos de video.

Cuando la línea BA se pone baja (*low*) quiere decir que el chip del video solicita tiempo al 6510. Entonces el otro chip dispone del tiempo suficiente para rematar la instrucción que esté realizando antes de que el VIC-II baje la línea de *control de habilitación de direcciones* (AEC) y desactive sin más los manejadores del bus de direcciones del 6510. Esto equivale a desarmar por sorpresa al 6510 con otra arma más poderosa: éste no se da cuenta de que ha sido puesto fuera de combate.

El tiempo robado del 6510 por el chip VIC-II es importante. Por ejemplo, las direcciones del puntero de caracteres deben ser tomadas después de cada octava línea de barrido visualizada en la pantalla (ya que los caracteres tienen ocho filas de pixels) y cada línea necesita 40 accesos consecutivos para tomar los punteros de la memoria de video (hay 40 caracteres por fila de pantalla). En el sistema PAL, el chip del video renueva una línea sí y otra no de las 625 líneas de barrido en la pantalla del televisor aproximadamente 25 veces por segundo; el sistema americano NSTC tiene 524 líneas, y las renueva 30 veces por segundo. Esto suma una buena cantidad de tiempo. De hecho, una sencilla operación aritmética muestra que el efecto neto es el freno en la velocidad del 6510 en el 15 o 20 % aproximadamente.

Los DMA no tienen ningún efecto notable sobre el funcionamiento interno de la máquina. Pero cuando está en cuestión el tiempo real de E/S estas "ausencias" imprevistas del 6510 pueden convertirse en un problema, por ejemplo, en el funcionamiento de la cassette. En tales casos puede ser necesario desconectar la pantalla (por medio de POKE 53265,11) durante la E/S, y volverla a conectar (con POKE 53265,27) después de finalizada la E/S. El que el empleo del acceso directo a la memoria que hace el VIC-II resulte problemático durante una E/S dependerá del dispositivo externo y del método empleado para comunicarse con él. En muchos casos, como el acceso a un disco, no será necesario desconectar la pantalla.

La primera interrupción de barrido conecta el modo en alta resolución



La segunda interrupción de barrido devuelve al modo de texto

Dimension Graphics

Las dos caras de la historia
Muchos juegos de aventuras comerciales escritos para el Commodore 64 emplean técnicas de interrupción de barrido para obtener dibujos en alta resolución y texto simultáneamente. *Spiderman*, cuyo autor es Scott Adams y que ilustramos aquí, utiliza el tercio superior de la pantalla para visualizar una posición en el juego; los restantes dos tercios se usan para describir la escena en el modo normal de visualización de textos. El explorador de barrido de la TV está programado para generar dos interrupciones cada vez que se inspecciona la pantalla: la primera en la parte superior y la segunda aproximadamente un tercio más abajo. Cada vez que sucede una interrupción de barrido se llama una rutina de interrupción, lo que hace que la visualización bascule entre los modos de alta resolución y el de texto

Partir la pantalla

Una característica interesante del Commodore 64 consiste en que, si se programa hábilmente, es posible dividir la pantalla visual en dos modos de resolución alta y baja. Esto puede hacer, por ejemplo, que un gráfico se esté visualizando al mismo tiempo que se imprime un texto o se realiza cualquiera otra interacción con el usuario en baja resolución sólo en una porción de la pantalla.

El programa para partir la pantalla que proporcionamos aquí muestra varios de los aspectos tratados. Se ha preparado una pequeña pantalla en alta resolución detrás de la ROM del intérprete de BASIC y se visualiza continuamente en el tercio superior de la pantalla. Al mismo tiempo, los dos tercios inferiores quedan en el modo normal de baja resolución.

Ya hemos estudiado el empleo de los vectores de la RAM en el Commodore 64. Mostramos cómo el cambio del valor de un puntero de dos bytes nos permitía desviar una rutina del sistema operativo a un código creado por nosotros. Para obtener una partición de la pantalla desviaremos la rutina IRQ de una manera algo sibilina.

Hay varios modos de disparar una IRQ en el 6510. La manera habitual es que uno de los temporizadores del CIA ponga un bit del registro del flag de interrupción (IFR) en la dirección 53273 ($D019) cada sesentavo de segundo. Esto provoca la ejecución de la rutina de servicio IRQ, la cual, entre otras cosas, inspecciona la matriz de conexiones del teclado para saber si se pulsó alguna tecla. También se ponen a uno los bits del IFR por medio del chip VIC-II cuando se cumplen ciertos eventos, tales como el caso de colisión de dos sprites o cuando el contador de barrido alcance un valor preestablecido. Hay otro registro, el *registro activador de interrupciones* (IER) en la dirección 53274 ($D01A) que actúa como un conector activador de la línea IRQ del 6510.

Si los bits correspondientes del IER son puestos por el programador a 1, entonces los bits activados del IFR dispararán una IRQ en el 6510.

La idea general del programa es la siguiente:

1) Cuando sucede una interrupción de barrido en la parte superior de la pantalla, activar el modo de mapa de bits y después establecer la interrupción

de barrido a mitad de la pantalla. Retorno de la interrupción (RTI).
2) A la interrupción en la mitad de la pantalla, volver a establecer el modo en baja resolución y establecer después la siguiente interrupción de barrido para que suceda en la parte superior de la pantalla (seguido de otro RTI).

Obsérvese un aspecto algo espinoso de esta idea: parte de la memoria normal de pantalla correspondiente al tercio superior de la pantalla se ha empleado para información del color en la pantalla de alta resolución: los dos tercios restantes se emplean para datos de caracteres de la manera habitual.

Sin embargo, el programa tiene una dificultad. Las interrupciones de barrido sucederán cada cincuentavo de segundo y deben atenderse inmediatamente, pues de lo contrario el barrido tendrá que seguir desplazándose, y visualizará datos erróneos. Ahora bien, en circunstancias normales, la rutina de servicio de IRQ se dispara cada sesentavo de segundo. La rutina de servicio es bastante larga y si ocurre justamente antes de la interrupción de barrido, se producirá un retraso antes de llegar a nuestra cuña de barrido del programa. Esto hay que evitarlo a toda costa, pues si no es así la pantalla basculará en la frontera de alta y la baja resolución.

La solución a este problema consiste en obligar a la rutina IRQ a que se ejecute inmediatamente después del código de la cuña de barrido. Esto se consigue desactivando el temporizador habitual del sesentavo de segundo, y así se desactiva el disparador habitual de IRQ, saltando a la rutina IRQ justamente después de haber ejecutado la cuña de barrido. Esto obliga a que los dos fragmentos de código se ejecuten en sincronía el uno con el otro. Naturalmente esto sólo significa que el teclado será marginalmente inspeccionado con menor frecuencia, lo que tiene el útil efecto lateral de acelerar algo la ejecución del BASIC, sin mayores daños.

Finalmente, hemos puesto la pantalla en alta resolución detrás de la ROM del intérprete de BASIC. Esto hace que el "puntearla" en BASIC resulte algo dificultoso, pues el lenguaje no puede leer el área de RAM que está detrás de la ROM del BASIC, y no podemos realizar los necesarios AND y OR para establecer los pixels individuales. No obstante, un POKE puede pasar por la ROM del BASIC y llegar hasta la RAM trasera, permitiendo el *plotting* simple desde el BASIC. En los programas de sólo lenguaje máquina no necesitamos la ROM del intérprete, y como no representan más que 8 K de espacio desperdiciado, se desconecta. Unos pequeños arreglos en el listado fuente le permitirán reposicionar la pantalla en alta resolución de modo que pueda usarse con mayor facilidad desde el BASIC.

Programa de partición de pantalla para C64

...rgador del BASIC

```
REM ********************************
REM **  CARGADOR BASIC DEL PROGRAMA  **
REM **       PARTICION PANTALLA 64   **
REM ** NOTA — EN ESTA VERSION       **
REM **  LA ALT. RES. ESTA BAJO EL BASIC **
REM **  LUEGO NO SE PUEDE USAR PEEK  **
REM **          EN SUBR. PLOT        **
REM ********************************

REM CARGA Y PRUEBA DE PARTICION PANTALLA **
DATA 76,50,192,49,234,173,25,208
DATA 240,37,169,255,160,21,141,25
DATA 208,173,0,221,73,2,141,0,221
DATA 141,17,208,73,32,141,17,208,41
DATA 240,4,169,121,160,255,141
DATA 208,140,24,208,108,3,192
DATA 169,41,254,141,14,220
DATA 169,3,192,173,21,3
DATA 141,20,3,169
DATA 173,17,208,41,95
DATA 141,18,208
DATA 208,169,1,141
DATA 133,1,169
DATA 133,248,160
DATA 200,208
DATA 169
DATA 162,4,169
DATA 251,230
DATA 133
REM CHECKSUM
REM INSERTA CODIGO
CHECKSUM ERROR":STOP
** PANTALLA **
REM CALCULA DIRECCION
REM TRAZA SEGMENTO
** CALCULO **
```

```
1410 U=INT(Y/8):V=INT(X/8)
1420 R=YAND7:C=XAND7
1430 M=CM + (40*U+V)*8
1440 RETURN
```

Listado assembly

```
;*************************************************
;*************************************************
;**              PARTICION PANTALLA 64          **
;*************************************************
;*************************************************

;
        *= $C000
        JMP START
;
        MEM1 = $F7
        MEM2 = $F9
        MEM3 = $FB
        SCN = $FD
        CINV = $314         ; VECTOR IRQ
        TEMP IRQ *=*+2
;
NEWIRQ =*                   ; NUEVO INICIO CUÑA IRQ
;
        LDA  $D019          ; MIRA EL ESTADO DE INTERRUPCION
        BEQ  NOTVIC         ; VINO LA IRQ DEL VIC?
        LDA  #$FF
        LDY  #$15
        STA  $D019          ; BORRA LATCH INT.
        LDA  $DD00
        EOR  #$02
        STA  $DD00          ; CAMBIA BANCOS
        LDA  $D011
        EOR  #$20           ; CAMBIA BIT DEL MODO MAPA DE BITS
        STA  $D011
        AND  #$20           ; COMPARA EL BIT BMM
        BEQ  LSCN           ; PARTE INFERIOR PANTALLA
        LDA  #$79           ; PONE EL BARRIDO EN LINEA 22
        LDY  #$FF
LSCN
        STA  $D012          ; ESTABLECE COMP. NUEVO BARRIDO
        STY  $D018
NOTVIC
        JMP  (TEMP IRQ)
;
START =*                    ; INICIALIZA CUÑA IRQ
;
        LDA  $DC0E
        AND  #$FE           ; DESACTIVA TEMPORIZADOR R
        STA  $DC0E          ; CIA #1
        LDA  CINV
        STA  TEMP IRQ       ;
        LDA  CINV+1         ; CAMBIA VECTOR IRQ
        STA  TEMP IRQ+1     ;
        LDA  #<NEWIRQ
        STA  CINV
        LDA  #>NEWIRQ
        STA  CINV+1
        LDA  $D011
        AND  #$5F           ; RESTABLECE MODO BMM
        STA  $D011
        LDA  #$00
        STA  $D012
        LDA  #$FF
        STA  $D019          ; RESTABLECE LATCHES DE IRQ
        LDA  #$01
        STA  $D01A          ; ACTIVA COMPARACION BARRID
        LDA  $01
        AND  #$FE
        STA  $01            ; DESCONECTA LA ROM DEL BASI
        LDA  #$00
        STA  MEM1           ; EST. PUNTS. P. 0 PARA APUNTA
        LDA  #$A0           ; INICIO AREA ALT. RES. DETRAS
        STA  MEM1+1         ; ROM DEL BASIC
        LDY  #$00
        LDX  #$1C
        LDA  #$00           ; BORRA ALT. RES.
DDD
        STA  (MEM1),Y
        INY
        BNE  DDD
        INC  MEM1+1
        DEX
        BNE  DDD
        LDA  #$BC
        STA  MEM1+1
        LDY  #$00
        LDX  #$04
        LDA  #$B7           ; ESTABLECE COLOR
DDE
        STA  (MEM1),Y
        INY
        BNE  DDE
        INC  MEM1+1
        DEX
        BNE  DDE
        LDA  $01
        DRA  #$01
        STA  $01            ; RESTAURA ROM BASIC
        RTS                 ; VUELTA AL BASIC
```

Héroe temerario

"Impossible mission" (Misión imposible), de Epyx, es un juego con excelentes gráficos y un guión con "carrera contra el tiempo" incluida

Impossible mission: Para el Commodore 64
Editado por: CBS (Columbia Broadcasting System) Software, 3-5 Rathbone Place, London W1, Gran Bretaña
Autor: Dennis Caswell
Palanca de mando: Necesaria
Formato: Disco o cassette

Entre los diversos tipos de programas de juegos, uno de los más populares es el juego de plataformas. En el mismo, el jugador (en el papel de héroe) debe obtener objetos que están situados en plataformas dentro de las diversas pantallas. Por lo general el héroe penetra en la pantalla por una de las esquinas inferiores y debe alcanzar las plataformas mediante una serie de ascensores, escaleras y trampolines (o lo que sea que haya puesto el programador a su disposición). A menudo, objetos vitales tales como llaves o un importante tesoro están colocados en una posición particularmente difícil que implica un considerable esfuerzo para determinar la mejor forma de llegar hasta ellos.

Los juegos de este tipo se complican aún más mediante la presencia de alienígenas u otras figuras gráficas peligrosas que se han de evitar, porque el mero hecho de tocarlas suele tener consecuencias fatales. Estos gráficos pueden ser estáticos o bien desplazarse siguiendo patrones preestablecidos. Además, muchos de estos sprites poseen capacidad de "disparo". Según cuáles sean las intenciones del programador, los movimientos de estos enemigos pueden ser más o menos inteligentes. Muchos de ellos se moverán hacia atrás o hacia adelante a lo largo de un único camino, mientras que otros pueden estar programados para "apuntar" hacia el héroe, limitando su libertad de acción.

No hay nada imposible
El objetivo de *Impossible mission* consiste en recorrer, mediante ascensores y pasillos, las diversas habitaciones de la guarida de un perverso científico. Tras entrar en las habitaciones, evitando a los robots, el jugador debe buscar entre los muebles partes de la contraseña que finalmente le permitirá entrar en el laboratorio del profesor Atombender. Ocasionalmente también podrá descubrir "somníferos", contraseñas que desconectan temporalmente a los robots

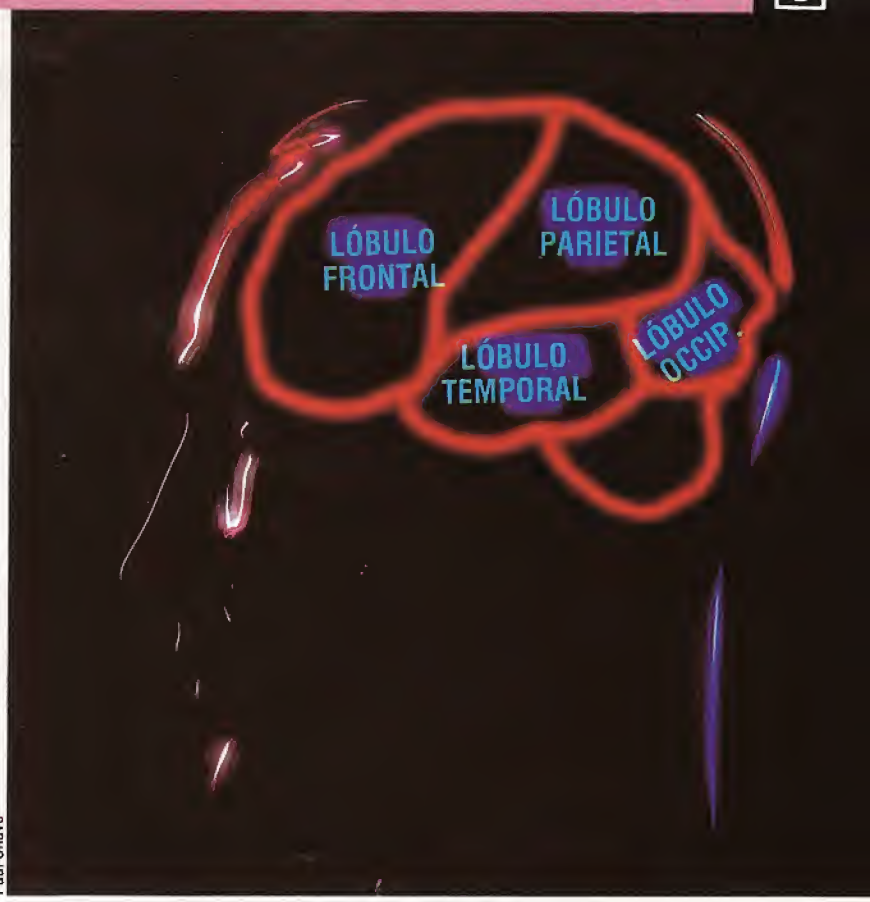
Los juegos de plataformas le ofrecen varias ventajas al programador. Entre ellas la más importante es el escaso código necesario para la implementación del juego. Una vez que el programador ha definido el diseño gráfico del héroe, los guardias, el tesoro, las plataformas, las escaleras y los ascensores, todas las diferentes "habitaciones" se pueden programar con sólo reposicionar los diversos elementos. Es obvio que ello supone un gran ahorro de espacio, que se puede utilizar para mejorar el nivel de detalle del juego o incrementar el número de habitaciones.

Impossible mission saca el máximo partido de la capacidad de los juegos de plataformas para implementar código de esta manera. El objetivo del juego consiste en encontrar los diversos trozos de una clave secreta (ocultos en el mobiliario de 32 habitaciones) que le permite al jugador irrumpir en la ciudadela del malvado científico Elvin. Los muebles están custodiados por robots y, ocasionalmente, por una inmensa bola negra. Algunos elementos del mobiliario contienen, asimismo, contraseñas especiales que, tras obtenerlas, le permiten al jugador volver a montar ascensores en las habitaciones y desconectar temporalmente los robots para poder pasar junto a ellos con toda seguridad. Estas contraseñas se entran a través de los terminales de ordenador situados en cada habitación. El jugador se desplaza de una habitación a otra mediante un ascensor y a través de una serie de pasillos. En la esquina inferior izquierda de la pantalla hay un mapa que muestra las habitaciones en las cuales se ha entrado hasta el momento y la posición actual del jugador.

Las contraseñas también se pueden obtener a partir de cuartos de codificación. Para recibir una contraseña, la habitación emite una cantidad de notas de altura variable, y el jugador debe acomodarlas por orden ascendente. Cuantas más contraseñas intente uno obtener, mayor será el número de notas que se toquen. En este sentido, *Impossible mission* es un juego de plataformas estándar. A diferencia de la mayor parte de los juegos de plataformas, el jugador no posee una cantidad de "vidas" establecida, sino que, en cambio, cada vez que el agente es "asesinado", incurre en una penalización de tiempo.

En este juego se le da un uso excelente al espacio de memoria ahorrado por los programadores. La característica más notable es la síntesis de voz. Los gráficos son, igualmente, muy buenos, y aunque las habitaciones no están repletas de ellos, los detalles individuales del agente, el mobiliario y los robots están dibujados con gran refinamiento. Todos estos elementos se conjugan para conformar un juego atractivo y muy bien trabajado.

En el cerebro



Paul Chave

Iniciamos una serie en la que trataremos en profundidad el apasionante tema de la inteligencia artificial y los importantes avances de la investigación en este campo

A principios de la década de los setenta, las investigaciones en el campo de la inteligencia artificial (*artificial intelligence*: AI) se hallaban estancadas. Se la consideraba, en líneas generales, como un aspecto marginal y extravagante de la ciencia informática. Por aquel entonces, en Gran Bretaña, un elaborado informe de sir James Lighthill para el Science Research Council recomendaba un drástico recorte de los fondos destinados a tal fin. En la actualidad, todo lo concerniente a la inteligencia artificial se encuentra en pleno auge, y los profesionales dedicados a su investigación se ven acosados por audaces inversionistas que les proponen tentadoras ofertas económicas.

Varias agencias gubernamentales europeas están financiando costosos programas de investigación y desarrollo por temor a quedarse rezagadas en la carrera por el desarrollo de la AI. Mientras, las firmas vendedoras de software están distribuyendo pomposos comunicados de prensa en los que redefinen sus productos como sistemas de inteligencia artificial.

Para comprender la situación en que se encuentra hoy la AI y la que es probable que alcance en el futuro, es útil, tal como sucede con tantas otras tecnologías, dar una mirada a su pasado. Podemos dividir nuestra condensada historia de la inteligencia artificial en cuatro períodos, cada uno de ellos de una década de duración y caracterizado por un tema dominante. Esto necesariamente supone una simplificación de las cosas, pero al hacerlo, se ponen de relieve los puntos principales. Podemos considerar que cada uno de los temas esenciales es la respuesta que con toda probabilidad habría obtenido uno de haberle preguntado a un investigador de AI de la época: "¿En qué consiste la inteligencia artificial?"

1950 Redes neurales
1960 Búsqueda heurística
1970 Sistemas expertos
1980 Aprendizaje de la máquina

En 1943, Warren McCulloch y Walter Pitts propusieron un modelo de neurona del cerebro humano y animal. Estas células nerviosas abstractas proporcionaron la base para una representación simbólica matemática de la actividad cerebral. Otros investigadores, en especial Norbert Wiener, elaboraron estas ideas junto con otras similares, dentro del mismo campo, que se dio en llamar "cibernética" (un sistema basado en la premisa de que se puede construir una máquina consciente utilizando como modelo métodos biológicos de realimentación y análisis). De la cibernética surgió, en la década de los cincuenta, la inteligencia artificial.

Los primeros investigadores de AI tomaron como su bloque constructivo la neurona formalizada de McCulloch. Considerando la inmensa complejidad del cerebro, no fue del todo sorprendente que no consiguieran generar sistemas inteligentes basados en este modelo. En efecto, ellos postulaban: "El cerebro es un solucionador inteligente de problemas, de modo que imitemos al cerebro." Pero el hardware de la época, por no hablar del software, no estaba a la altura de la tarea.

Uno de los pocos sistemas que obtuvo cierto éxito en aquellos días fue el Perceptron de Rosenblatt. Se trataba de un sistema visual elemental al cual se le podía enseñar a reconocer patrones. Como se puede apreciar en el diagrama (p. 1703), el Perceptron se compone de una cuadrícula finita de células sensibles a la luz que conforman una retina en miniatura. Además hay varios elementos detectores de configuración (denominados gráfica-

Mentes mecánicas
La inteligencia artificial trata del diseño de sistemas de ordenador que lleven a cabo tareas que, de realizarlas un ser humano, requerirían inteligencia. Sin embargo, su esfera de acción se ha ampliado para incluir funciones de percepción, tales como ver y oír. La principal preocupación de la investigación sobre AI es programar máquinas de modo que imiten aspectos del comportamiento y la comprensión humanos

mente demonios) que controlan el estado de grupos de células de la cuadrícula. Cuando hay presentes subpatrones característicos, responden enviándole una señal a un elaborador de decisiones. Éste multiplica cada señal proveniente de un demonio local por un factor de valor relativo positivo o negativo, y se suman los números resultantes. Si el total supera un umbral establecido, un Perceptron puede distinguir entre dos clases de imágenes, si bien se pueden ampliar los mismos principios para tratar con más de dos.

En un momento dado hubo grandes esperanzas de que el Perceptron pudiera abordar una amplia gama de tareas de resolución de problemas, pero éstas se disiparon enseguida. Los estudiosos de la AI comenzaron a considerar el pensamiento humano como una coordinación de tareas esencialmente simples relacionadas con la manipulación de símbolos. Si bien esto constituyó ostensiblemente un gran cambio en la dirección de las investigaciones, los diseñadores se hallaban al menos en terreno firme, puesto que los ordenadores podían hacer cosas tales como efectuar búsquedas, comparar símbolos, etc., que ellos identificaban con los fundamentos de la solución inteligente de problemas. Lo difícil consistía en unir entre sí estas actividades simples.

Los estudiosos más influyentes de la década de los sesenta fueron Alan Newell y Herbert Simon, de la Universidad de Carnegie-Mellon, que, entre otras investigaciones, trabajaron en demostración de teoremas y ajedrez por ordenador. Su logro más impresionante fue un programa denominado GPS (*General Problem Solver*: solucionador general de problemas). Éste era general en el sentido de que el usuario definía un "entorno de tarea" en función de los objetos de un dominio particular y los operadores que se podían aplicar a esos objetos. Sin embargo, esta generalidad se ceñía a puzzles con un juego relativamente pequeño de estados y reglas bien definidos. Podía trabajar con las torres de Hanoi, criptoaritmética y otros tipos de problemas similares en los que microcosmos formalizados representaban los parámetros dentro de los cuales se podían realmente resolver problemas. Lo que no podía hacer el GPS era resolver lo que la gente consideraría problemas del mundo real, como realizar diagnósticos médicos o tomar decisiones de gestión basadas en monedas fluctuantes.

El GPS se sustentaba en la idea de que la solución del problema implicaba una búsqueda a través de un espacio de soluciones potenciales. Para que la búsqueda fuera eficaz debía estar dirigida por reglas heurísticas (que aprenden a partir de sus propios descubrimientos) que la conducían hasta el destino deseado, hallando el camino esencialmente a través del método de ensayo y error. Por consiguiente, un robot que deambulara a través de un laberinto tendría que utilizar una técnica de búsqueda exhaustiva si no supiera nada sobre la estructura de ese laberinto; pero si tuviera alguna forma de saber cuándo se estaba "acercando", se podía esperar que llegara a su estado objetivo más rápidamente (pero no siempre, porque no existen garantías de que la heurística funcione, y en ocasiones puede conducir a callejones sin salida). Durante este período, los estudiosos de la AI idearon varias estrategias de búsqueda guiadas heurísticamente.

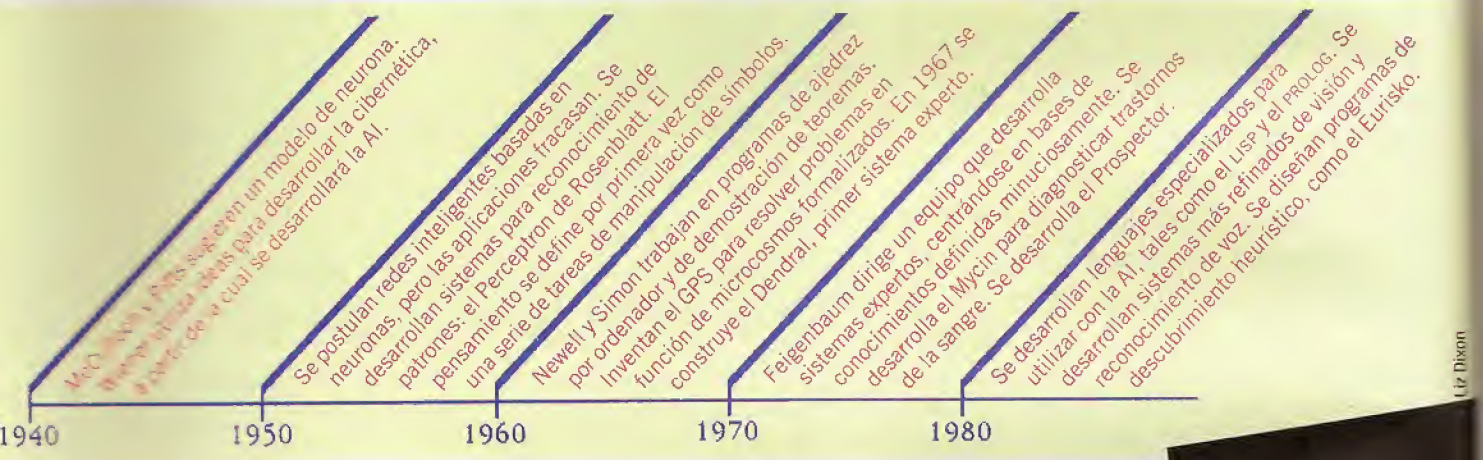
El GPS, como hemos dicho, no era muy eficaz para resolver problemas de la vida real. En los años setenta un equipo dirigido por Edward Feigenbaum, de la Universidad de Stanford, comenzó a remediar ese defecto. En vez de intentar informatizar la inteligencia general, se centraron en áreas muy concretas de pericia. Y así nació el sistema experto.

El primer sistema experto fue el Dendral, un intérprete de espectrograma de masa construido ya en 1967, si bien el más influyente resultaría ser el Mycin, que data de 1974. El Mycin diagnostica infecciones bacterianas de la sangre y receta la medicación terapéutica. Ha dado lugar a toda una familia de "clones" para diagnóstico médico, algunos de los cuales son de uso clínico común. Por ejemplo, Puff, una herramienta para diagnóstico de función pulmonar basada en el plan Mycin, es de uso común en el Pacific Medical Center, situado en las cercanías de San Francisco.

El Mycin introdujo varias características nuevas que se han convertido en los rasgos distintivos de los sistemas expertos. En primer lugar, su "conocimiento" consiste en cientos de reglas como ésta:

REGLA N.º 47:
IF
1) la ubicación del cultivo es sangre, y
2) la identidad del organismo no se conoce con certeza, y
3) la tinción del organismo es gramnegativa, y
4) la morfología del organismo es de bastoncito, y
5) el paciente ha sufrido graves quemaduras

Perspectiva histórica
La inteligencia artificial se está convirtiendo en un área de investigación práctica cada vez más importante, con aplicaciones en muchos campos. La historia de la AI, sin embargo, abarca un período muy breve en comparación con muchos otros campos científicos, como se puede apreciar en el diagrama inferior...




Cortesía de Computer Weekly

Liz Dixon

Edward Feigenbaum
Como director de un equipo de AI de la Universidad de Stanford (California), Edward Feigenbaum desarrolló los primeros sistemas expertos. Éstos fueron notables porque, apartándose de la idea de programas de inteligencia general, se constituyeron en sistemas que operaban dentro de límites de conocimientos y objetivos definidos con suma precisión

THEN existen evidencias que sugieren débilmente (0,4) que la identidad del organismo es seudomomas.

En segundo lugar, estas reglas son probabilísticas. Shortliffe, el inventor del Mycin, que era doctor en medicina, desarrolló un esquema basado en factores de certidumbre para permitir que el sistema llegara a conclusiones verosímiles a partir de una evidencia incierta. Por consiguiente, el número 0,4 no es estrictamente una probabilidad; es, esencialmente, un "factor de falsedad". El punto significativo es, sin embargo, que el Mycin y sistemas similares pueden llegar a conclusiones correctas aun con información incompleta y parcialmente errónea. Emplean un método de razonamiento aproximado (basado ya sea en probabilidades, *fuzzy logic*, factores de certidumbre o algún otro cálculo de probabilidad) para producir una buena estimación de la verdad a partir de un conjunto imperfecto de datos.

En tercer lugar, el Mycin puede explicar su propio proceso de razonamiento. El médico que lo utiliza puede interrogarlo de diversas maneras, ya sea para preguntarle cómo llegó a una determinada conclusión o bien por qué está solicitando una cierta clase de información. El sistema responde desandando y describiendo el proceso deductivo que condujo al estado actual. Este nivel de amabilidad hacia el usuario fue esencialmente un derivado del estilo de programación basado en reglas.

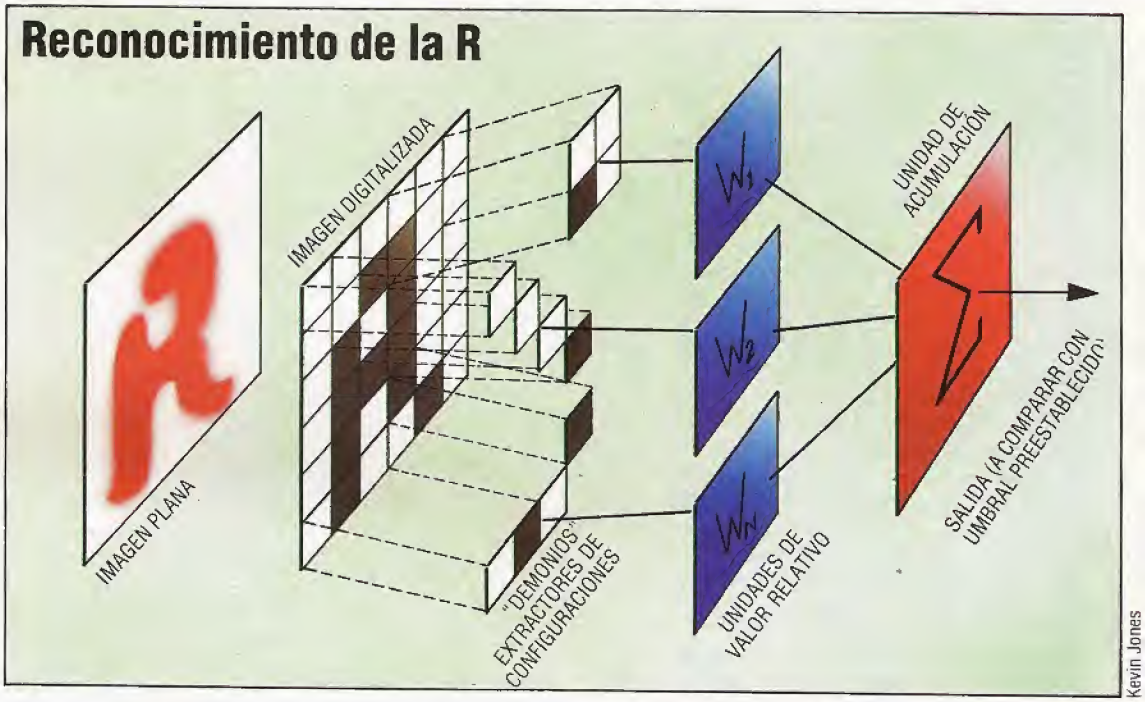
El factor último y esencial es que el Mycin *funciona*. Ejecuta aquello que un ser humano sólo puede hacer tras años de entrenamiento. En realidad, el Mycin se utiliza más para la enseñanza que para el diagnóstico, pero lo cierto es que está despertando un creciente interés entre las grandes corporaciones, los gobiernos y los *mass media*.

Y así llegamos a los años ochenta. Los sistemas expertos están de moda y su "ingrediente activo" es el conocimiento, porque el alcance y la calidad de su base de conocimientos determina el éxito de un sistema experto. Pero el conocimiento no es algo que uno pueda comprimir en un programa de ordenador como si se tratara de la pasta dentífrica de un tubo. Codificar la pericia de un experto humano puede ser un proceso largo y laborioso. De modo que mientras el mundo se maravilla ante los sistemas expertos, la inteligencia artificial ha pasado a concentrarse en el problema del aprendizaje de la máquina, que es una forma de adquirir conocimiento de forma automática. La AI siempre ha sido un objetivo móvil y ahora mismo en el centro de ese objetivo hay un programa que se denomina Eurisko. Se trata de un programa de descubrimiento que amplía y perfecciona su propio cuerpo de reglas heurísticas automáticamente, por inducción. Aparte de ganar durante tres años consecutivos el juego de guerra naval *Trillion credit squadron* (a pesar de que se intentó impedirlo mediante cambios en el reglamento), el Eurisko también se ha aplicado a problemas prácticos. Un resultado fue la invención de una nueva puerta lógica tridimensional en el campo del diseño de circuitos integrados. Existen pocas dudas en el sentido de que sistemas como el Eurisko representan la vanguardia de la investigación en AI. Y puesto que la inteligencia artificial constituye la vanguardia de la ciencia informática, es aquí donde se han de buscar los indicios para el futuro de ésta.

No deja de ser irónico que, al volver a concentrarse en el aprendizaje, las investigaciones en el campo de la inteligencia artificial hayan regresado a sus raíces, porque en los primeros días de la cibernética el aprendizaje se consideraba el problema clave.

Esta serie de capítulos representa una guía práctica a la inteligencia artificial. A lo largo de la misma iremos cubriendo las principales áreas problemáticas de la AI (como el proceso de la visión y del lenguaje natural), así como las técnicas que se han desarrollado para abordarlas. Como es habitual, todos los programas que se ofrezcan a modo de ejemplo estarán en BASIC, y en los recuadros de *Complementos al* BASIC nos referiremos al BBC Micro, el Commodore 64 y el Spectrum.

Las percepciones del Perceptron
La imagen a muestrear (en este caso, la letra R) se proyecta sobre un plano y se digitaliza. Los *demonios* muestran un pequeño grupo de pixels (p. ej., 4 por vez) y responden si está presente el patrón para cuyo reconocimiento han sido programados. La respuesta de cada demonio (0 o 1) se multiplica por un factor de valor relativo, según cuál sea su importancia dentro del patrón como un todo, y se suman entre sí las respuestas. Este resultado se compara entonces con un valor umbral. Si es mayor que el umbral, entonces el sistema reconoce la forma original; de lo contrario, no la reconoce. El umbral y los valores relativos de los demonios se pueden regular cuando se entrena al Perceptron para que reconozca una forma determinada



La clasificación adecuada

Al ir de tiendas en busca de un administrador de bases de datos, ¿cuáles son los puntos principales que hay que tener en cuenta?

Cuando se utiliza un procesador de textos, el tiempo que consume la búsqueda de archivos o el desplazamiento de bloques de caracteres (lo que equivale a la clasificación) es un factor enojoso. Sin embargo, tras haberlas llevado a cabo, estas actividades por lo general no vuelven a ser necesarias. El esfuerzo de "búsqueda y clasificación" es, por tanto, un porcentaje relativamente pequeño del tiempo total de ejecución del programa. Es la entrada de datos (digitar el texto) y la impresión lo que tienden a consumir la mayor cantidad de tiempo.

En el caso de los DBM, sin embargo, esta situación se invierte. En términos generales, los registros son "documentos" muy pequeños cuya entrada demanda poco tiempo. Cuando se utiliza un DBM, éste invierte la mayor parte de su tiempo clasificando los registros y buscando a través de ellos para extraer los que se requieren. Si el archivo de datos es mayor que la capacidad total de la RAM, como es lo más probable, se requerirá una exhaustiva lectura y escritura en un soporte de almacenamiento masivo. Las deficiencias del almacenamiento en cassette, lento y secuencial, se harán evidentes de inmediato. A menos que su base de datos sea limitada en tamaño y en importancia (el catálogo de una pequeña colección de discos, p. ej.), usted deberá considerar el tener que invertir en un sistema basado en disco.

La mayoría de las bases de datos se emplean en un contexto de gestión de uno u otro tipo, en los que rige el lema *time is money* (el tiempo es dinero). En otras palabras, si necesita un DBM, precisará un sistema basado en disco, aunque, mientras tanto, el *Archive* en los microdrives del QL constituye una alternativa adecuada.

¿Cuántos registros ha de ser capaz de manipular mi DBM? ¿Vale la pena tener uno con más registros de los que realmente necesito?

Su aplicación de base de datos puede o no exigir la capacidad de manipular gran número de registros. Pero intente calcular el número máximo que podrá necesitar alguna vez, duplíquelo, y después busque un DBM capaz de manipular esa cantidad. Si su base de datos es para control de stock e inventario y no cuenta con muchos componentes, puede que no necesite un DBM capaz de manipular 32 000 registros. Pero si está utilizando el DBM para catalogar una biblioteca, quizá 32 000 registros no sean suficientes. Los DBM capaces de manejar más de 64 000 registros son bastante raros y, de todos modos, normalmente requerirían potentes ordenadores con memorias muy grandes en discos rígidos.

Si no estoy muy seguro respeto a la cantidad de campos que necesitaré, ¿cómo puedo decidir qué DBM comprar?

La mayoría de los DBM permiten que cada registro contenga sólo una cierta cantidad de campos. A menudo resulta difícil saber de antemano cuántos campos se requerirán. La mayoría permiten un número de campos más que suficiente, si bien la longitud de éstos a menudo se limita a una línea, por lo cual puede resultar complicada la utilización de campos de textos de más de una línea. Una limitación más común que la cantidad de campos permitida es el número de caracteres admisibles por registro. Una cifra típica podría ser 1 020 caracteres por registro, permitiendo al usuario elegir cómo dividir esta cantidad entre el número de campos y la longitud de caracteres de cada campo. En cualquier caso, asegúrese de que el número máximo de caracteres por campo y de campos por registro permita que usted construya los registros que desea.

Los campos clave ¿son un aspecto importante? De ser así, ¿cuántos necesitaré?

Un campo clave es aquel mediante el cual el DBM puede efectuar una búsqueda u otra manipulación. Si un campo no está designado como clave, no puede ser utilizado para extraer registros. He aquí dos ejemplos de cómo emplear los campos clave típicos:

```
SEARCH PROVEEDOR FOR 'Timson Engineering Ltd'
```

Se está buscando en el campo clave PROVEEDOR de cada registro una pareja que posea la serie de caracteres especificados, o:

```
SEARCH PRECIO>=45.50
```

Mediante el campo clave PRECIO se están buscando cifras mayores o iguales que 45.50. Algunos DBM permiten especificar todos los campos, o cualquier número de ellos, como campos clave cuando se crea el "esqueleto" del registro; otros sólo permiten otorgarles tal designación a un cierto número de campos. Como método práctico, cuantos más campos se puedan designar como campos clave, tanto mejor.

¿Me veré limitado al lenguaje que se suministre con mi ordenador, o podré programar yo mismo el DBM, prescindiendo del mismo?

Algunos DBM incluyen un lenguaje de programación incorporado que permite efectuar de forma automática sofisticadas secuencias de actividades. En un entorno de gestión, con estos lenguajes es posible escribir programas que le ahorran al operador una gran cantidad de trabajo.

Dos ejemplos notables son el *dBase II* y el *Archive*. Si sus requerimientos para base de datos tienen formas establecidas de tratar los datos, tales como "imprimir todas las ventas del día seguidas por una lista de todos los componentes en stock y una lista de todos los componentes que se encuentran por debajo del nivel de 'nuevo pedido'", entonces lo que se debe buscar es un DBM con lenguaje de programación incorporado. Si, por el contrario, no sabe cómo se utilizará la base de datos día a día, entonces bastarán las instrucciones usuales de "lenguaje de interrogación".

¿Qué he de buscar si los registros que crearé habrán de relacionarse con otros datos diferentes a los de los registros?

Los administradores de bases de datos más simples pueden trabajar con un solo archivo de registros a un tiempo. Para muchas aplicaciones esto es perfectamente adecuado. No obstante, es mucho más útil la capacidad de un DBM para trabajar con dos o más archivos al mismo tiempo. La situación clásica en la que resulta útil es la de una base de datos de control de stock en la cual cada componente puede tener más de un proveedor. Tendríamos un archivo de base de datos para "componentes" y otro archivo separado para "proveedores". Si usted efectúa una interrogación en un archivo, es sumamente útil poder efectuar interrogaciones a otro archivo, estando ambos relacionados.

Para ilustrar este punto, supongamos que llevamos una tienda de antigüedades y tenemos un archivo de base de datos "stock" sobre lo que tenemos en existencia. También podríamos llevar un registro de nuestros proveedores en el cual apuntaríamos las cosas que ellos tienen disponibles pero que aún no hemos comprado. Nuestros registros para "MOBILIARIO, SUECO" podría no tener entradas, pero una referencia cruzada a su archivo PROVEEDORES podría revelar a KURT JAKOBSEN ANTIK, GAMLA STAN 56, ESTOCOLMO. Su registro podría revelar:

```
DESCRIPCION:        SOFA
FABRICANTE:         ANDERSEN
FECHA:              1824
PRECIO COR. SUEC.:  12000
```

DESCRIPCION:	SOFA
FABRICANTE:	GRIMM
FECHA:	1874
PRECIO COR. SUEC.:	9800

e infinidad de otras entradas de artículos disponibles. En otras palabras, cuando no basta un único archivo, la referencia a uno o más archivos diferentes puede ser lo que hace falta.

Teóricamente, siempre es posible tener suficientes campos libres dentro de cualquier archivo determinado para permitir la adición de cualquier información que pudiera requerirse. En la práctica, puede que ello no sea posible. Supongamos que el señor Jakobsen nos telefonea para decirnos que hoy mismo acaba de recibir dos encantadores sofás del s. XIX que quizá podrían interesarnos. Obviamente, es mucho más sencillo entrar los detalles en el registro de JAKOBSEN que pasar por todos los campos DESCRIPCION del archivo STOCK y entrar las nuevas adquisiciones del señor Jakobsen.

Para simplificar, siempre que usted tenga una situación en la que hay más de una relación de uno a uno entre partes de un registro de base de datos con otros datos, lo que uno debe considerar fundamentalmente es un DBM de archivos múltiples.

Los campos dependientes y calculados parecen ser bastante útiles. ¿De qué forma me pueden ser de ayuda en el trabajo que realizaré con el DBM?

Algunos DBM ofrecen un refinamiento que se denomina *campos dependientes*. Éstos son campos que aparecen (durante la entrada de datos y posteriormente) sólo si son necesarios. Por ejemplo, una base de datos que posea un campo N. de HIJOS podría no visualizar campos extras si la entrada fuera 0, y NOMBRE DEL PRIMER HIJO, EDAD DEL PRIMER HIJO, etc., si la entrada fuera 1, y así sucesivamente.

Los campos dependientes son, no obstante, un mero refinamiento y muy raramente son esenciales. Algunos DBM también permiten usar los datos de ciertos campos como argumentos en operaciones aritméticas, casi como si se tratara de una minihoja electrónica. Tales sistemas permitirían, por ejemplo, multiplicar la cantidad de artículos de un campo N. EN STOCK por la cifra del campo PRECIO en cada registro, así como sumar el total de los resultados de los registros en un archivo para dar un valor total en resultado de stock. Este tipo de facilidad casi siempre viene unida a un lenguaje de programación incorporado y puede ser muy útil.

¿Cuáles son las diferencias entre los DBM activados por menú y los activados por instrucciones, y es alguno de ellos más adecuado que el otro?

Un DBM activado por menú es aquel que presenta opciones en la pantalla en cada una de las etapas de actividad. Un programa activado por instrucciones espera que usted aprenda su vocabulario de instrucciones para hacerlo funcionar (que a menudo implica una combinación tecla Control-más-letra de pulsaciones de tecla). Aunque los programas activados por menú tienden a ser más fáciles para el principiante, los activados por instrucciones son generalmente más rápidos de usar una vez que se han aprendido las instrucciones. Siempre y cuando las instrucciones se elijan de forma lógica (S para SEARCH, P para PRINT, etc.), es probable que, a la larga, el software activado por instrucciones sea más fácil de usar.

¿Qué clase de facilidades tienen incorporadas los DBM para verificar errores e impedir entradas erróneas?

Un DBM bien diseñado le permitirá especificar qué tipo de datos se pueden entrar en cada campo, lo que contribuye, por tanto, a reducir las entradas erróneas. Por ejemplo, 31/02/85 sería rechazada automáticamente, como lo sería una entrada como:

PRECIO: reborde para montar medidor de aceite

El mejor software siempre incluye la facilidad de imponer límites sobre el tipo de datos admisible; por otra parte, en ocasiones la verificación excesivamente rígida puede ser una incomodidad. En cualquier caso, compruebe las facilidades que se ofrecen para validación de datos.

Habiendo dicho lo anterior, ¿qué DBM debo comprar?

Recomendar la mejor compra para un DBM es imposible; usted tiene que saber para qué lo utilizará antes de adquirir uno. No obstante, nuestro consejo sería, para todas las aplicaciones excepto las menos sofisticadas, un DBM basado en disco, con acceso a archivos múltiples y que, además del lenguaje de interrogación normal, poseyera un lenguaje de programación incorporado.

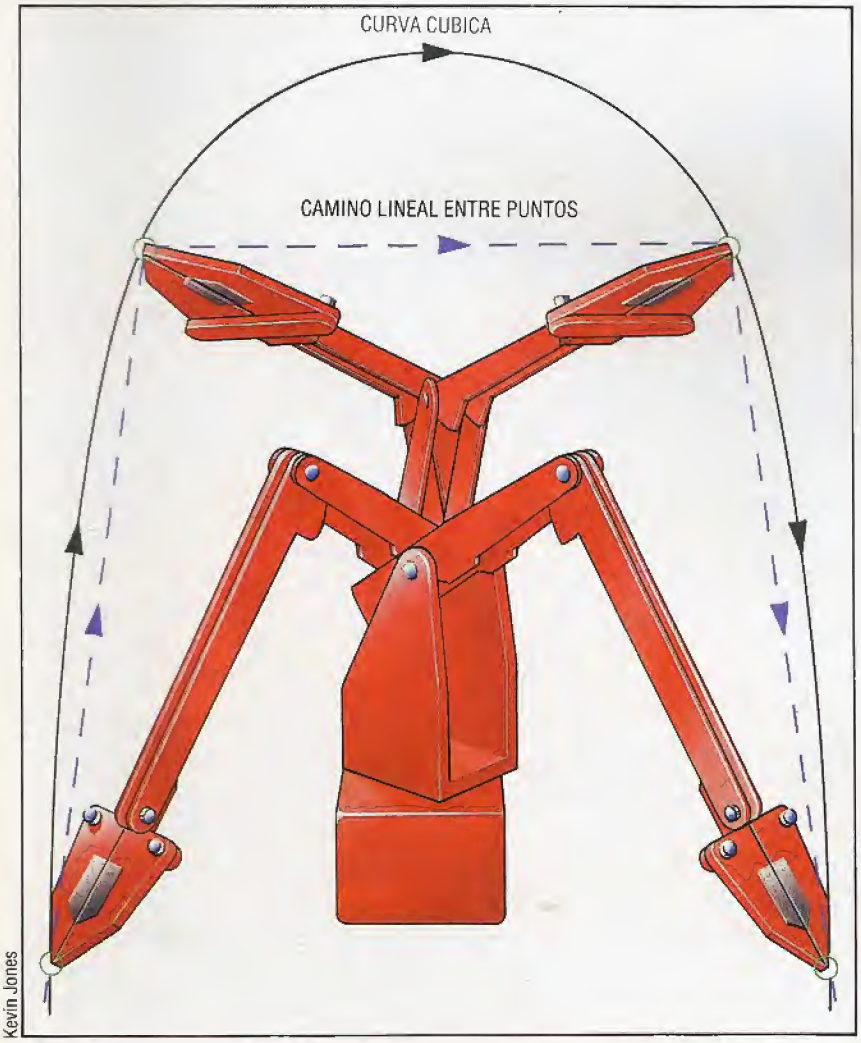
La curva de recorrido

En este último capítulo de nuestro proyecto revisaremos el software que hemos desarrollado hasta ahora

Suavizando los bordes salientes
Si cuando se utiliza el programador para el brazo se guardan cuatro posiciones clave para éste, entonces, al reproducir la secuencia, el brazo se desplazará linealmente entre los puntos (indicado mediante la línea de puntos). Sin embargo, se pueden calcular varios puntos intermedios que generen una curva uniforme (conocida como *curva cúbica*) que pase a través de las cuatro posiciones guardadas. Las coordenadas de los puntos de esta curva se pueden utilizar para hacer que el brazo efectúe un barrido más uniforme a lo largo de su camino programado

La rutina moverservo desarrollada como parte del software controlador del brazo reduce la torpeza inherente del brazo cuando se mueve desde una posición a otra. Los motores reciben su información angular desde una serie de posiciones de la memoria, que comienzan en ANGULO.

Para mejorar la uniformidad del movimiento del brazo, cuando los motores asumen nuevas posiciones en respuesta a cambios producidos en las posiciones ANGULO, los valores no se colocan (POKE) directamente en estas posiciones, sino en un grupo de posiciones correspondientes etiquetado NUEVAPOS. Al ser llamada, la rutina moverservo compara los valores de las posiciones correspondientes de ANGULO y NUEVAPOS e incrementa o decrementa ANGULO de la forma apropiada, de una en una unidad. Repite el proceso hasta que los valores de NUEVAPOS y ANGULO son los mismos.



Kevin Jones

Al colocar (POKE) los valores de ángulo desde BASIC en NUEVAPOS en vez de en ANGULO, efectivamente aislamos los motores contra cambios de ángulo súbitos y de gran envergadura. La tarea de desplazar un carrete de hilo de coser desde una posición conocida hasta una taza es ahora relativamente sencilla. Se puede guiar el brazo lentamente a través de los movimientos, utilizando las teclas adecuadas del teclado, y las posiciones clave de la curva se pueden guardar en una matriz. La información de la matriz se puede reproducir a cualquier velocidad insertando un factor de demora en moverservo y recorriendo paso a paso la matriz.

Es imposible introducir más mejoras. Para una primera aproximación, todos los movimientos angulares son proporcionales a los cambios de NUEVAPOS, según a qué distancia respecto al pivote se ajuste cada enlace, en comparación con el radio de la palanca activadora del motor. Por consiguiente, la rutina moverservo hace mover cada parte del brazo de forma uniforme y con una velocidad angular constante (los grados de rotación por segundo) hasta su nueva posición. Sin embargo, cada vez que alcance una nueva posición, puede ser que el brazo haya de cambiar repentinamente la dirección al iniciar su camino hacia la siguiente posición guardada en la secuencia; en otras palabras, es posible que alcance una discontinuidad.

Este problema no es tan grave como el problema original de efectuar grandes cambios súbitos de ángulo. No obstante, puede ser evitado guardando una cantidad de puntos intermedios en lugar de introducir cambios radicales entre una posición guardada y la siguiente.

Con frecuencia se puede mejorar aún más el movimiento del brazo mediante el empleo de un software más sofisticado, pero ha de lograrse el equilibrio entre velocidad y espacio. Si el carrete de hilo se mueve especificando solamente cuatro posiciones separadas (la primera cuando se recoge el carrete, la segunda cuando se lo levanta, la tercera al asumir una nueva posición encima de la taza, y la cuarta cuando se baja el carrete a la taza) se utiliza poca memoria, pero se producen grandes discontinuidades.

Una forma de reducirlas consistiría en especificar, pongamos por caso, 60 posiciones intermedias, en cuyo caso el movimiento sería comparativamente fluido y elegante. Pero al hacer esto, el almacenamiento de los datos consumiría sustancialmente más memoria, por no mencionar el tiempo que llevaría entrar manualmente todas las posiciones.

La alternativa exige dedicar tiempo a calcular una cantidad de valores intermedios y después combinar las posiciones guardadas manualmente con las posiciones de conexión calculadas. Ahora se nos plantea el problema de cómo calcular estos valores intermedios.

Si consideramos cuatro puntos especificados, entonces se podría disponer una banda articulada de acero, para que tocara tres puntos cualesquiera de los cuatro en cualquier momento dado, en especial si los dos extremos estuvieran en posiciones convenientes. El equivalente matemático de disponer un trozo de acero articulado consiste en interpolar

puntos intermedios mediante el empleo de una *regla flexible cúbica* (así llamada porque incluye valores elevados a la tercera potencia).

El programa que ofrecemos aquí demuestra cómo se pueden interpolar los puntos intermedios que conforman una curva cúbica a partir de los cuatro puntos especificados. El movimiento resultante entre los puntos será una curva de barrido uniforme: una mejora considerable respecto a la interpolación lineal. Para usar este método con un sistema de cuatro motores, como el de nuestro brazo-robot, será necesario interpolar matemáticamente una regla flexible cúbica para cada motor.

Reglas flexibles cúbicas

BBC Micro:

```
1000 REM ********************************
1010 REM **        CURVAS CUBICAS BBC        **
1020 REM ********************************
1030 :
1040 MODE 0
1050 VDU 23,240,24,126,66,219,219,66,126,24
1060 V=7
1070 DIM X(V+3), Y(V+3), M(V+3), Z(V+3)
1080 X(2)=0: X(3)=10 :X(4)=40 :X(5)=50
1090 Y(2)=2 :Y(3)=12 :Y(4)=12 :Y(5)=2
1100 REM ampliar los extremos linealmente
1110 X(1)=X(2)+(X(3)-X(2))
1120 Y(1)=Y(2)-(Y(3)-Y(2))
1130 X(6)=X(5)+(X(5)-X(4))
1140 X(7)=X(6)+(X(6)-X(5))
1150 Y(6)=Y(5)+(Y(5)-Y(4))
1160 Y(7)=Y(6)+(Y(6)-Y(5))
1170 GLG:VDU 5:MOVE 100,700
1180 PRINT"Demostracion de una rutina"
1190 PRINT"que incorpora una curva cubica"
1200 PRINT"que une puntos"
1210 FOR I=2 TO 5
1220 MOVE X(I)*20-4,Y(I)*20+15:PRINT CHR$(240);
1230 NEXT
1240 GOSUB 1500:REM ESTABLECER AKIMA
1250 FOR X=X(2) TO X(5)
1260 GOSUB 1330:REM FUNCION AKIMA
1270 IF X=X(2) THEN MOVE X*20,Y*20
1280 IF X<>X(2) THEN DRAW X*20,Y*20
1290 NEXT
1300 END
1310 :
1320 :
1330 REM **** FUNCION AKIMA ****
1340 REM AKIMA RUCKDESCHEL LIBRO 2 (BYTE/MCGRAW-HILL)
1350 N=1
1360 REM COMPROBAR SI X SE HALLA EN EL RANGO DE LA TABLA
1370 IF X<X(1) OR X>X(V-2) THEN RETURN
1380 REM HALLAR INTERVALO DE TABLA PERTINENTE
1390 I=0
1400 I=I+1:IF X>=X(I) THEN 1400
1410 I=I-1
1420 REM COMENZAR INTERPOLACION
1430 B=X(I+1)-X(I)
1440 A=X-X(I)
1450 Y=Y(I)+Z(I)*A+(3*M(I+2)-2*Z(I)-Z(I+1))*A*A/B
1460 Y=Y+(Z(I)+z(I+1)-2*M(I+2))*A*A*A/(B*B)
1470 RETURN
1480 :
1490 :
1500 REM **** ESTABLECER AKIMA ****
1510 REM CALCULAR COEFICIENTES AKIMA
1520 FOR I=1 TO V-1
1530 REM PASAR I A I+2
1540 M(I+2)=(Y(I+1)-Y(I))/(X(I+1)-X(I))
1550 NEXT I
1560 M(V+2)=2*M(V+1)-M(V)
1570 M(V+3)=2*M(V+2)-M(V+1)
1580 M(2)=2*M(3)-M(4)
1590 M(1)=2*M(2)-M(3)
1600 FOR I=1 TO V
1610 A=ABS(M(I+3)-M(I+2))
1620 B=ABS(M(I+1)-M(I))
1630 IF A+B<>0 THEN Z(I)=(A*M(I+1)+B*M(I+2))/(A+B)
1640 IF A+B=0 THEN Z(I)=(M(I+2)+M(I+1))/2
1650 NEXT I
1660 RETURN
```

Commodore 64:

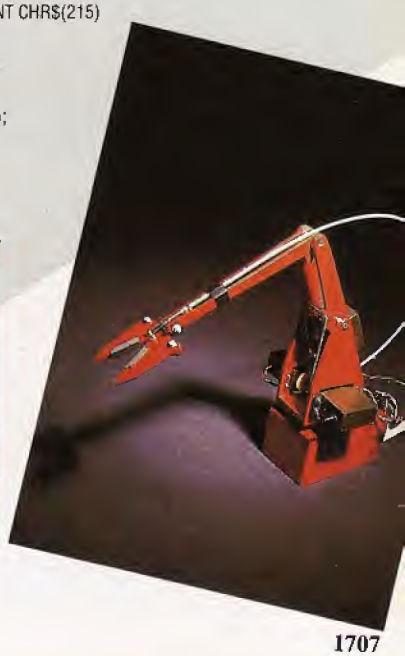
```
10 REM ******************************
20 REM **     CURVAS CUBICAS CBM     **
30 REM ******************************
40 :
50 V=7:FOR I=1 TO 25:DW$=DW$+CHR$(17):NEXT
60 DIM X(V+3),Y(V+3),M(V+3),Z(V+3)
70 X(2)=0:X(3)=5:X(4)=20:X(5)=25
80 Y(2)=2:Y(3)=12:Y(4)=12:Y(5)=2
90 REM ** EXTENDER LOS EXTREMOS LINEALMENTE **
100 X(1)=X(2)-(X(3)-X(2))
110 Y(1)=Y(2)-(Y(3)-Y(2))
120 X(6)=X(5)+(X(5)-X(4))
130 X(7)=X(6)+(X(6)-X(5))
140 Y(6)=Y(5)+(Y(5)-Y(4))
150 Y(7)=Y(6)+(Y(6)-Y(5))
160 PRINT CHR$(147)
170 PRINT"DEMOSTRACION DE UNA RUTINA QUE"
180 PRINT"INCORPORA UNA CURVA CUBICA"
190 PRINT"QUE UNE PUNTOS"
200 GOSUB 3000:REM DIBUJAR PUNTOS
230 GOSUB 1000:REM ESTABLECER AKIMA
235 J=2
240 FOR X=X(2) TO X(5)
250 GOSUB 2000:REM FUNCION AKIMA
255 IF X=X(J) THEN J=J+1:GOTO 270:REM NO DIBUJAR
260 N=X:M=Y:GOSUB 5000:PRINT CHR$(46)
270 NEXT
280 GET A$:IF A$<>"" THEN 280
290 END
1000 REM **** ESTABLECER AKIMA ****
1010 REM CALC COEFICIENTES AKIMA
1020 FOR I=1 TO V-1
1030 M(I+2)=(Y(I+1)-Y(I))/(X(I+1)-X(I))
1040 NEXT I
1050 M(V+2)=2*M(V+1)-M(V)
1060 M(V+3)=2*M(V+2)-M(V+1)
1070 M(2)=2*M(3)-M(4)
1080 M(1)=2*M(2)-M(3)
1090 FOR I=1 TO V
1100 A=ABS(M(I+3)-M(I+2))
1110 B=ABS(M(I+1)-M(1))
1120 IF A+B<>0 THEN Z(I)=(A*M(I+1)+B*M(I+2))/(A+B)
1130 IF A+B=0 THEN Z(I)=(M(I+2)+M(I+1))/2
1140 NEXT I
1150 RETURN
2000 REM **** FUNCION AKIMA ****
2010 N=1
2020 IF X<X(1) OR X>X(V-2) THEN RETURN:REM COMPROBAR RANGO
2030 I=0
2040 I=I+1:IF X>=X(I) THEN 2040
2050 I=I-1
2060 REM COMENZAR INTERPOLACION
2070 B=X(I+1)-X(I)
2080 A=X-X(I)
2090 Y=Y(I)+Z(I)*A+(3*M(I+2)-2*Z(I)-Z(I+1))*A*A/B
2100 Y=Y+Z(I)+Z(I+1)-2*M(I+2))*A*A*A/(B*B)
2110 RETURN
3000 REM **** IMPRIMIR PUNTOS ****
3010 FOR I=2 TO 5
3020 N=X(I):M=Y(I):GOSUB 5000:PRINT CHR$(215)
3030 NEXT I
3040 RETURN
5000 REM **** POSICION EN N,M ****
5010 PRINT CHR$(19);
5020 PRINT TAB(N);LEFT$(DW$,25-M);
5030 RETURN
```

Complementos al BASIC

Spectrum:

Suprimir las líneas 1040 y 1050 del listado para el BBC Micro e insertar las siguientes:

```
1170 CLS
1220 CIRCLE X(I)*5-4,Y(I)*5+15,10
1270 IF X=X(2) THEN PLOT X*5,Y*5
1280 IF X<>X(2) THEN DRAW X*5,Y*5
```



Serpiente en el C64

Este simpático juego es también un buen ejercicio de programación para crear juegos. He aquí la versión para el Commodore 64

En este juego, usted es una serpiente que se desplaza contoneándose por la pantalla. El cambio de dirección se consigue pulsando cualquier tecla. Para poder desplazarse es preciso que se alimente. Felizmente se halla rodeado por cantidad de setas. Pero, ¡cuidado! Si bien las setas azules son excelentes, ha de evitar las negras, puesto que son venenosas. Cada seta azul le proporciona las calorías necesarias para avanzar diez líneas. ¡Procure no morirse de hambre pero sin acabar envenenado!

```
  5 REM ************************************
 10 REM *              SERPIENTE             *
 15 REM ************************************
 20 GOSUB 1000
100 GET X$
110 IF X$<>"" THEN D=-D
120 T=T+D
130 IF T<L0 THEN T=L0
140 IF T>L1 THEN T=L1
150 T1=T+40
160 IF PEEK(T1)=65 THEN 500
170 IF PEEK(T1)=88 THEN S=S+10:H=H+10
180 PRINT B$;
190 P=INT(RND(TI)*40)
200 POKE 1984+P,88
210 POKE 1984+P+M,BC
220 P=INT(RND(TI)*40)
230 IF RND(TI)<0.5 THEN 260
240 POKE 1984+P,65
250 POKE 1984+P+M,RC
260 POKE T,CT
270 POKE T+M,TC
280 S=S-1
290 IF S=0 THEN 500
300 H=H+1
310 GOTO 100
500 PRINT B$;
510 POKE T,CT
520 POKE T+M,TC
530 FOR I=1 TO 500
540 NEXT I
550 IF H>RE THEN RE=H
560 PRINT CHR$(19);
580 FOR I=1 TO 10
590 PRINT B$;
600 NEXT I
610 PRINT TAB(13)"PUNTOS[1SPC]:";H;
    CHR$(19);
620 FOR I=1 TO 15
630 PRINT B$;
640 NEXT I
650 PRINT TAB(10)"PUNTUACION[1SPC]
    MAXIMA[1SPC]:";RE;CHR$(19);
660 FOR I=1 TO 20
670 PRINT B$;
690 GET X$
700 NEXT I
710 PRINT TAB(13)"OTRA[1SPC]?";
720 GET X$
730 IF X$="" THEN 720
740 IF X$<>"N" THEN 20
750 END
1000 PRINT CHR$(147)
1010 POKE 53280,5
1020 POKE 53281,13
1030 CT=19
1040 TC=9
1050 D=-1
1060 T=1404
1070 T1=T
1080 BC=6
1090 RC=0
1100 S=100
1110 M=54272
1120 B$=CHR$(17)
1130 L1=1423
1140 L0=1384
1150 H=0
1200 FOR I=0 TO 24
1210 PRINT B$;
1220 NEXT I
1230 PRINT CHR$(144);
1240 RETURN
```

Un lugar en el sol

En esta ocasión examinaremos el Wren Executive, sólido ordenador portátil construido por Thorn EMI, que cuenta con un modem y una pantalla monocromática incorporados

El ordenador Wren Executive ha tenido una historia azarosa. Construido por Thorn EMI, originalmente fue comercializado por Prism. Cuando a comienzos de 1985 la empresa quebró, pareció que el Wren iba a convertirse en otra víctima de la turbulenta industria informática. No obstante, recientemente Opus Supplies ha asumido la distribución del ordenador y tiene planes para volver a lanzar la máquina. En este capítulo analizaremos las posibilidades de que dispone el equipo para hacerse un lugar en el enormemente competitivo mercado de gestión.

El Wren Executive está diseñado para ser una máquina "portátil". Se trata, en consecuencia, de un conjunto autocontenido que consta de un ordenador, una pantalla y unidades de disco. Aunque a primera vista la carcasa parece ser de metal pintado, en realidad está construida casi por completo en plástico grabado, siendo de metal sólo la base y los contornos del teclado. En la parte posterior de la máquina hay un asa para transportarla. Al igual que muchos ordenadores portátiles, quizá el ordenador sea un poco pesado como para poder transportarlo durante mucho rato, y tal vez se describiría mejor como un ordenador de mesa que se puede transportar de un área de trabajo a otra.

El teclado posee la configuración QWERTY usual, con la única excepción de que los símbolos * y ≠ poseen sus propias teclas. A la izquierda de las teclas de máquina de escribir hay cinco teclas de función programables que, utilizadas conjuntamente con las teclas Control o Shift, pueden cumplir hasta 15 funciones distintas. Estas teclas poseen finalidades diferentes dependiendo de la aplicación que se esté empleando, pero básicamente se utilizan para entradas de palabras clave individuales de instrucciones CP/M, tales como PIP y RENAME. En el lado opuesto del teclado se hallan las cuatro teclas del cursor y una tecla Home para llevar el cursor a la esquina superior izquierda de la pantalla. Ésta mide 150 por 105 mm y es la típica pantalla monocromática que se suministra con máquinas de gestión, con una resolución para textos de 80 por 25 caracteres. La única diferencia notoria es que el color de primer plano es naranja en lugar del verde habitual. Esto contribuye a una visualización brillante y atractiva, muy apropiada para largas sesiones de trabajo. A la derecha de la pantalla hay un par de unidades de disco flexible de 5 ¼ pulgadas.

Chris Stevens

Máquina insólita
Distribuido por Opus Supplies, el ordenador Wren Executive es una máquina portátil económica, que posee la característica insólita de ejecutar tanto BASIC BBC como el sistema operativo CP/M. Otras características incluyen unidades de disco gemelas, pantalla monocromática y un modem incorporado

La pantalla tiene el inconveniente del diseño de la carcasa. El Wren se asienta sobre el escritorio en una posición bastante baja, sin patas para poder inclinar la máquina. La pantalla está empotrada en la carcasa hacia atrás, con un reborde de 40 mm que se proyecta hacia fuera de la pantalla. Aunque el reborde está situado en ángulo, quizá encuentre que éste oscurece la parte superior de la pantalla si usted es más bien alto o suele sentarse cerca del teclado. Ésta no es una dificultad seria, pero puede restar comodidad.

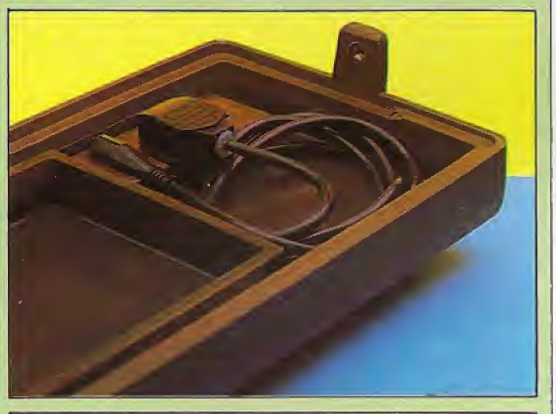
El Wren está bien provisto de interfaces. Hay varias puertas en la parte posterior que permiten conectar el ordenador a una amplia gama de periféricos. A diferencia de muchas máquinas de gestión de precio similar, el Wren posee un modem de llamada automática incorporado. El usuario del Wren está capacitado, por lo tanto, para acceder a las bases de datos más importantes, como Prestel y Micronet, sin tener que adquirir para ello ningún equipo adicional (aunque, por supuesto, usted habrá de suscribirse a una de las bases de datos antes de que se le permita establecer conexión).

El Wren viene incluso con un cable para conectarlo a la red telefónica. El software para comunicaciones que se entrega con el ordenador permite almacenar varios números de teléfono, cuyo marcaje se realiza de forma automática cuando se desea.

Junto al conector para comunicaciones hay un par de puertas que posibilitan la instalación de dispositivos de control externos, como un ratón y palancas de mando. A continuación de estas puertas hay un conector D de 25 vías que proporciona la interface para comunicaciones seriales RS232C. Los conectores RS232 se utilizan normalmente en ordenadores de gestión para modems externos.

Abundancia de interfaces
Una de las mejores características del Wren es la riqueza de interfaces con que cuenta la máquina de forma estándar. Éstas incluyen no sólo las interfaces convencionales Centronics y RS232C, sino también un enchufe telefónico para el modem (incluyendo un cable). Asimismo, hay un conector en paralelo para facilitar la instalación de un disco rígido Winchester

Sólo hay que enchufarla
Cuando se traslada la máquina, la pantalla y las unidades de disco se protegen mediante una cubierta de plástico pesado que se coloca en la parte delantera de la máquina. Aplicando el lema de que "sólo hay que enchufarla", los fabricantes han dado el inusual paso de proporcionar con el enchufe que viene ya instalado un cable de potencia. Cuando se transporta la máquina, este cable se almacena en el interior de la carcasa



A la perfección
Al estar construido por Thorn EMI, parece natural que se haya optado por empaquetar en la máquina su propia serie de software de aplicaciones: *Perfect writer*, *Perfect filer* y *Perfect calc*. También se incluye un programa general llamado *Executive desktop*, software para comunicaciones y BASIC BBC

Obviamente, ello no es necesario en el Wren, de modo que será primordialmente usado para redes de área local e impresoras en serie.

A la izquierda de la puerta RS232 hay una interface en paralelo Winchester en la cual se puede conectar un sistema de almacenamiento en disco rígido. La adición de un sistema de disco Winchester puede añadir al sistema entre 500 Kbytes y 50 megabytes de almacenamiento adicional. Asimismo, hay una interface en paralelo compatible con Centronics que proporciona el medio para imprimir en modo *hard copy*. Por último, en el extremo derecho, hay un conector para pantalla en color RGB. Entre las interfaces para impresora y pantalla hay un diminuto botón de reset que le proporciona al sistema un arranque en frío.

El Wren está basado en el procesador de ocho bits Z80B. Éste quizá sea el punto más débil del ordenador. En una época en que los fabricantes de micros personales compiten duramente para desarrollar micros de 16 bits, el Wren parece algo anticuado en su proceso. Aunque moderadamente rápido de acuerdo a las pautas de ocho bits, no existe

Chris Stevens

Pantalla
El Wren dispone de una pantalla monocromática incorporada brillante y fácil de leer

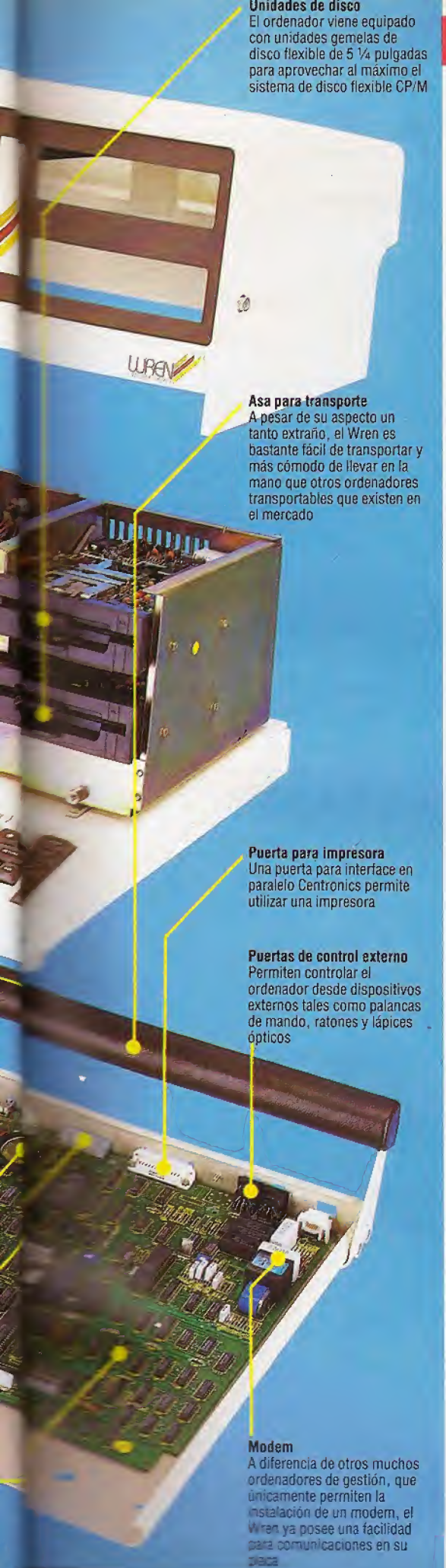
Puerta RS232C
A través de este conector se proporciona comunicación directa con otros ordenadores y otras aplicaciones en serie

Conector para pantalla en color
Para el caso de que el usuario desee sacar partido de las facilidades de color del Wren, éste posee una puerta adicional para monitor RGB, conectable a una pantalla externa

Altavoz
Las facilidades de sonido las proporciona el propio altavoz del Wren

Interface Winchester
En esta interface en paralelo se puede enchufar un disco rígido Winchester, que aumenta notablemente la capacidad de almacenamiento externo del ordenador

Chips RAM
El Wren Executive puede direccionar hasta 400 K de RAM a través del proceso de conmutación de bancos de memoria



comparación posible con las velocidades de micros de 16 bits tales como el IBM PC o el Apple Macintosh.

Al estar basado en el procesador Z80, era natural que los diseñadores del Wren optaran por utilizar como sistema operativo el CP/M. El problema de esto es que, según los estándares recientes, el CP/M es, a pesar de su potencia, un sistema operativo más bien hostil. Los diseñadores han atenuado este problema mediante la incorporación en el sistema operativo de un procedimiento activado por menú que permite que el usuario elija una aplicación mediante un cursor y la pulsación de RETURN. El sistema solicitará entonces la colocación del disco adecuado y ejecutará automáticamente la aplicación. Por consiguiente, muchos usuarios no se verán en la necesidad de emplear el CP/M directamente, puesto que casi todo lo que necesita el usuario de gestión medio se le proporciona a través del menú.

Los discos de aplicaciones que vienen empaquetados con el ordenador son el disco de sistema CP/M, el *Perfect writer* (procesador de textos), el *Executive desktop* (paquete para llevar hora, fechas, agenda y citas), la base de datos *Perfect filer*, la hoja electrónica *Perfect calc* y un disco que contiene el software para comunicaciones y el BASIC.

El BASIC que se utiliza para programar al Wren resulta ser, sorprendentemente, el BASIC BBC estándar. Si bien la máquina posee una implementación casi completa del lenguaje, el hecho de estar basado en el procesador Z80 y no en el 6502 ha supuesto algunas diferencias, principalmente en las rutinas de entrada/salida. Asimismo, la implementación del lenguaje parece haber generado algunos errores. Por ejemplo, en las modalidades 0 y 1 el cursor desaparece de la vista. La tecla Delete no funciona en estas modalidades, y hacer retroceder el cursor e intentar reescribir provoca que el nuevo carácter se sobreimprima sobre el antiguo. Ello impide toda programación seria en estas modalidades, si bien las otras parecen funcionar perfectamente.

Expectativas inmediatas

En última instancia, el futuro éxito del Wren Executive dependerá de que pueda obtener un lugar adecuado en el mercado. Con un modem incorporado, el sistema operativo CP/M y el BASIC BBC a un precio muy competitivo, en el momento de su lanzamiento, a comienzos de 1984, el Wren parecía una oferta excepcional. Desde entonces el potencial de ventas para máquinas de gestión de ocho bits, incluso en el extremo inferior del mercado, ha venido disminuyendo rápidamente. Así y todo, la máquina sigue siendo demasiado cara como para asestar un buen golpe en el mercado del ordenador personal. Y esto es especialmente cierto ahora que Amstrad ha implementado el CP/M en su gama de micros personales.

Queda, entonces, el mercado educativo, en el cual parece ser que el Wren causará su mayor impacto. El hecho de que la máquina utilice BASIC BBC, que está muy difundido en las escuelas, junto con un modem incorporado para aprovechar las diversas bases de datos escolares que se están creando, podría hacer que la máquina les resultara atractiva a los establecimientos educativos que intentan darle a su dinero la mejor aplicación.

WREN EXECUTIVE

DIMENSIONES

440×410×250 mm

CPU

Z80B, operando a 6 MHz

PANTALLA

Pantalla para textos de 80×25, pantalla para gráficos de 640×256

INTERFACES

Puerta para pantalla, interface Centronics, puerta RS232C, enchufe telefónico, puertas gemelas para control externo, interface para disco Winchester

LENGUAJES DISPONIBLES

BASIC BBC

TECLADO

57 teclas de máquina de escribir, cinco teclas de función programables, cinco teclas para control del cursor

DOCUMENTACION

Además del manual del usuario, cada una de las aplicaciones que se proporcionan posee su propio manual. La documentación está bien organizada y ofrece al usuario explicaciones completas

VENTAJAS

La cantidad de interfaces, junto con un modem incorporado y los numerosos programas incluidos, hacen de esta máquina una excelente inversión

DESVENTAJAS

La tecnología de 8 bits empleada en el Wren se está volviendo cada vez más anticuada. Asimismo, parece haber errores en algunas de las aplicaciones

Rebelión a bordo

Existe la posibilidad de que la tripulación se amotine, se apodere del barco y emprenda el regreso, acabando, de este modo, con nuestro juego mercantil "El Nuevo Mundo"

Existen varias contingencias mayores y menores que influyen en el viaje de forma positiva o negativa. Éstas se pueden seleccionar al azar durante cada semana para su ejecución, pero no se volverán a producir tras haberse seleccionado una vez.

Son ocho los factores que pueden desencadenar un motín. A pesar de que el barco tiene capacidad para una tripulación compuesta por 16 personas, toda cantidad por encima de 12 hará que se lo considere repleto y las condiciones de hacinamiento pondrán descontenta a la tripulación. Si no se contrató un cocinero, o si éste falleció durante el viaje, la tripulación habrá de llevar a cabo tareas de cocina, lo que reducirá la calidad de la comida ¡y creará mayor conflictividad!

Dado que avistar un albatros es un buen presagio, la tripulación quedará relativamente satisfecha; pero si es abatido, lo que trae mala suerte, aumentarán las probabilidades de que se produzca el motín. Poner a la tripulación a media ración de alguna de las provisiones contribuirá aún más al descontento de la misma, y algo similar ocurrirá si las reservas de capital del barco se vuelven inferiores a los salarios adeudados. Por último, en el momento de su contratación se le aseguró a la tripulación que el viaje duraría ocho semanas. Toda dilación de este lapso provocará descontento entre los tripulantes, que se intensificará con cada semana adicional que se requiera para concluir el viaje.

Para comprobar si las condiciones son suficientes para iniciar una rebelión se crea un factor de amotinamiento, MF. Cada condición se comprueba al comienzo de una semana llamando desde el bucle principal del programa a una rutina de amotinamiento. Si el resultado de cada comprobación es positivo, se le suma un valor al factor de amotinamiento. Este procedimiento continúa hasta que MF llega a 100, en cuyo momento tendrá lugar el motín. Hay, asimismo, un factor de demora, de hasta 30, que se incluye en la evaluación semanal del factor de amotinamiento.

La línea 879 envía al programa principal a la subrutina de la línea 7200, que efectúa una comprobación semanal del factor de amotinamiento MF, que al comienzo de la rutina se establece en 0. Si en cualquier etapa del viaje se pone a la tripulación a media ración, la variable H$ se establecerá en S. Ya que obtener los alimentos suficientes es una consideración de tanta importancia para la tripulación, darles medias raciones de cualquier provisión esta-

Módulo 10: Un motín

Rutina de amotinamiento

```
879 GOSUB 7200
```

Adición al bucle principal del viaje

```
7200 REM MOTIN
7210 MF=0
7215 IF H$="S"THEN MF=MF+30
7220 NC=0
7225 FOR T=1 TO 16
7228 IF TS(T,1)=5 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN
     NC=1:T=16
7230 NEXT
7235 IF NC=0 THEN MF=MF+30
7240 IF A$="S"THEN MF=MF-20
7245 IF B$="S" THEN MF=MF+30
7250 IF CN>12 THEN MF=MF+30
7255 IF WT>M0 THEN MF=MF+30
7260 IF WK>8 THEN MF=MF+((WK-8)*10)
7275 MF=MF+INT(RND(1)*30)
7280 IF MF<75 THEN RETURN
7282 PRINT CHR$(147)
7284 IF MF>100 THEN 7300
7285 S$="LA SITUACION EN EL BARCO*":GOSUB 9100
7286 S$="ESTA EMPEORANDO*":GOSUB 9100
7287 S$="Y ALGUNOS DE LOS TRIPULANTES*":GOSUB 9100
7288 S$="HABLAN YA DE MOTIN!*":GOSUB 9100
7290 PRINT:GOSUB 9200
7292 S$=K$:GOSUB 9100
7294 GET I$:IF I$=""THEN 7294
7299 RETURN
7300 PRINT CHR$(147)
7305 PRINT:GOSUB 9200
7310 S$="LA TRIPULACION SE HA AMOTINADO*":GOSUB 9100
7312 S$="PORQUE:*":GOSUB 9100
7313 X=0
7314 IF H$<>"S" THEN 7320
7315 GOSUB 9200:X=X+1:PRINT X;
7316 S$="HAN ESTADO A MEDIA RACION*":GOSUB 9100
7318 S$=" DURANTE PARTE DEL VIAJE*":GOSUB 9100
7320 IF NC<>0 THEN 7325
7321 GOSUB 9200:X=X+1:PRINT X;
7322 S$="NO HAY COCINERO*":GOSUB 9100
7324 S$=" Y LA COMIDA ES ASQUEROSA*":GOSUB 9100
7325 IF B$<>"S" THEN 7330
7326 GOSUB 9200:X=X+1:PRINT X;
7327 S$="EL ALBATROS FUE ABATIDO!*":GOSUB 9100
7330 IF CN<13 THEN 7335
7331 GOSUB 9200:X=X+1:PRINT X;
7332 S$="LA TRIPULACION ESTA HACINADA*":GOSUB 9100
7335 IF M0>=WT THEN 7340
7336 GOSUB 9200:X=X+1:PRINT X;
7337 S$="NO HAY SUFICIENTE ORO*":GOSUB 9100
7338 S$=" PARA PAGARLES SUS SALARIOS*":GOSUB 9100
7340 IF WK<=8 THEN 7350
7341 GOSUB 9200:X=X+1:PRINT X;
7342 S$="LLEVAN NAVEGANDO*":GOSUB 9100
7343 S$=" MAS DE 8 SEMANAS*":GOSUB 9100
7350 PRINT:GOSUB 9200
7360 S$="LA TRIPULACION SE APODERA DEL BARCO*":GOSUB 9100
7362 GOSUB 9200
7363 S$="Y EMPRENDE EL REGRESO*":GOSUB 9100
7370 GOSUB 9200
7372 S$="DEJANDOTE A TI ABANDONADO*":GOSUB 9100
7373 S$="EN UN BOTE A LA DERIVA*":GOSUB 9100
7374 S$="OJALA QUE ALGUIEN TE RECOJA":GOSUB 9100
7375 PRINT:GOSUB 9200
7380 PRINT"      FIN DEL JUEGO"
7382 END
```

Complementos al BASIC

Spectrum:
Introduzca las siguientes modificaciones:

```
7282 CLS
7294 LET I$=INKEY$:IF I$=""THEN GO TO 7294
7300 CLS
```

BBC Micro:
Introduzca las siguientes modificaciones:

```
7282 CLS
7294 I$=GET$
7300 CLS
```

blecerá H$ en S para el resto del viaje. H$ se comprueba mediante la rutina de amotinamiento, sumándole 30 a MF si H$ es S. La línea 7215 comprueba H$ e incrementa MF si fuera necesario.

La rutina de amotinamiento comprueba entonces si hay algún cocinero a bordo estableciendo la variable NC en 0, y prepara un bucle de 1 a 16, revisando el primer elemento de la matriz de fortaleza/categoría de la tripulación TS(,), en busca de un 5, que representaría un cocinero. Si la fortaleza del cocinero no es ni 0 ni −999, lo que implicaría la presencia de un cocinero sano, NC se establece en 1. Si no se contrató ningún cocinero, o si el mismo hubiera muerto durante el viaje, NC permanece en 0. La línea 7235 comprueba NC y, si NC es igual a 0, le suma 30 al factor de amotinamiento.

Si durante la travesía ya se hubiera avistado el albatros, entonces A$ se habría establecido en S en la línea 6055 de la subrutina del albatros. Ello trae buena suerte, puesto que en la línea 7240 se le quita 20 al factor de amotinamiento. Si uno ha derribado al albatros, la línea 6162 habrá establecido B$ en S, lo que trae mala suerte, y la línea 7245 incrementa en 30 el factor de amotinamiento. La línea 7250 comprueba si el número de tripulantes es mayor que 12 y, de ser así, le suma 30 al factor de amotinamiento, haciéndose eco de este modo de los sentimientos generales de la tripulación respecto a las condiciones de hacinamiento.

La factura del total de salarios está representada por WT y el dinero que queda en las arcas del capitán mediante MO. La línea 7255 comprueba si la factura de salarios es mayor que el dinero que queda en el pozo de reserva y, de ser así, le suma otros 30 a MF. El factor de amotinamiento se incrementa en 10 cada semana adicional del viaje tras las ocho primeras semanas. La línea 7260 comprueba si la travesía ya ha durado ocho semanas y, si así fuera, la ecuación le resta 8 al número de semanas (WK), multiplica el resto por 10 y le suma este resultado a MF. En la línea 7275, se le añade a MF un factor al azar entre 0 y 29.

Si, tras comprobar todas las condiciones, el total para el factor de amotinamiento es menor que 74, la línea 7280 devuelve el control al programa principal. Si el factor es mayor que 100, la línea 7284 envía el programa a 7300, en cuyo punto se produce el motín. Si el factor se halla comprendido entre 75 y 100, las líneas 7285-7288 advierten al jugador que la situación en el barco está empeorando y que entre la tripulación se habla de amotinamiento, antes de devolver el control al programa principal.

Si la tripulación se rebela, el programa determina las causas e imprime el motivo. La línea 7314 comprueba si H$ es igual a S e informa al jugador si la tripulación ha estado a media ración durante parte del viaje. La línea 7320 comprueba si NC es igual a 0, lo que significaría que no hay ningún cocinero a bordo, y se le dirá al jugador que el motín fue consecuencia de la falta de competencia culinaria. La línea 7325 determina si el albatros fue abatido o no y, en caso afirmativo, le informa al jugador que fue un factor que contribuyó al amotinamiento. La línea 7330 comprueba si el barco estaba repleto, la 7335 determina si hay suficiente dinero para pagar a la tripulación y la 7340 examina la duración del viaje. Por último, se informará que algunos tripulantes han sido muertos y que los amotinados se han apoderado del barco y han emprendido el regreso.

Factores del motín

Hay ocho factores que contribuyen a crear descontento entre la tripulación y que se utilizan para generar un factor de amotinamiento, MF, para cada semana del viaje. Si los ocho factores combinados hacen que el factor de amotinamiento sea mayor que 100, entonces se produce un motín y el capitán del barco es abandonado en un bote a la deriva

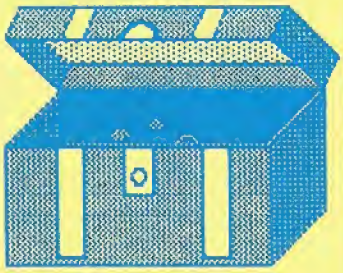
H$="S": medias raciones

NC=0: no hay cocinero

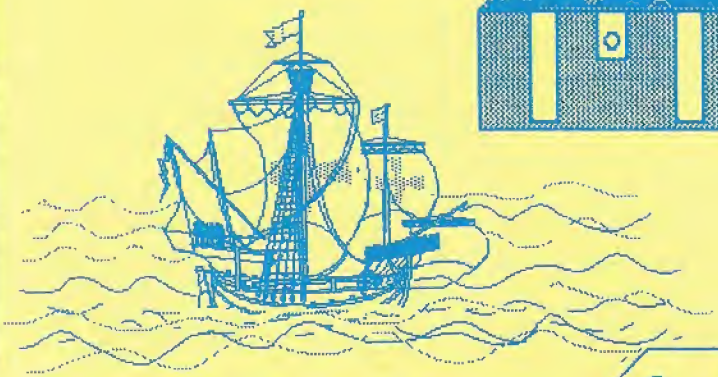
A$="S": avistado albatros

B$="S": abatido albatros

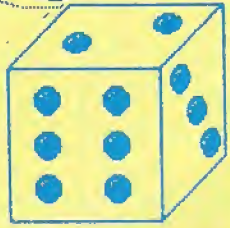
CN>12: hacinamiento



WT>MO: salarios superan arcas



WK>8: navegando durante>8 semanas



factor aleatorio

Ian McKinnell

Dúo dinámico

Analizaremos conceptos que nos permitirán procesar archivos de tamaño no especificado

El PASCAL ofrece dos procedimientos estándares (new y dispose) y un tipo especial de datos que, conjuntamente, proporcionan un poder y flexibilidad extraordinarios a la *asignación* y *desasignación dinámica* de memoria. Un tipo *puntero* es sencillamente una variable que, en vez de contener un valor de datos tal como un entero, "apunta a" un entero o cualquier otro objeto de datos, estructurado o no. La notación que se utiliza es similar a la empleada cuando denotamos un buffer *de siguiente dirección* de archivos (que, en efecto, apunta al siguiente registro del archivo, si bien de una forma totalmente diferente). Un tipo puntero se define colocando delante del identificador (del tipo de datos hacia el cual deseamos apuntar) una flecha hacia arriba:

```
TYPE
  MatrizLarga     = ARRAY 1..100 OF real;
  indirecta       = ↑MatrizLarga;
```

Todos los compiladores de ISO PASCAL soportan el mismo símbolo alternativo ("@"). La definición de tipo reserva una única posición de memoria que se utilizará para retener la dirección de una matriz grande sólo cuando se la cree mediante el procedimiento new. En el ínterin, el valor del puntero permanece sin definir, tal como sucedería con cualquier otra variable, de modo que:

```
VAR
  direccion       : indirecta;
  numero          : integer;
```

reservaría espacio para una dirección de máquina (16 bits en un micro de ocho bits) y un entero, sin estar ninguno de ellos inicializado en ningún valor determinado.

Así como desearíamos inicializar el entero a cero antes de utilizarlo, al puntero se le podría asignar específicamente el valor especial NIL. Ésta es una palabra reservada del PASCAL, que significa tan sólo que el puntero no señala hacia ningún lugar útil, y es el equivalente del valor numérico cero (lo que significa la ausencia de cualquier número verdadero). Por consiguiente, las dos variables del ejemplo se podrían asignar así:

```
direccion   :=NIL
numero      :=0;
```

Puesto que NIL es un valor constante que pertenece a un tipo genérico, quizá hubiera sido preferible definirlo como un identificador en el lenguaje. Resulta obvio que, en este sentido, Wirth cambió de parecer, porque en MODULA−2 la misma palabra es, en realidad, un identificador predefinido, no una palabra reservada. Para demostrar el empleo de new y dispose, por razones de simplicidad sólo utilizaremos enteros.

Cuando se necesita una variable de puntero para señalar un elemento nuevo, llamamos al procedimiento new del PASCAL, que asigna espacio para el elemento y coloca su dirección en el puntero. Después se alude al elemento de datos "desreferenciando" al puntero, p, mediante la notación p↑. Observe que ahora la fecha hacia arriba va detrás del identificador de puntero, al igual que en la notación buffer-archivo, y se podría imaginar como "el elemento hacia el cual apunta p". Huelga decir que es un error desreferenciar un puntero que no esté definido o que no apunte a ninguna parte.

Tras acabar con todos los datos creados mediante punteros (new), se puede llamar al procedimiento dispose del PASCAL. Éste es el opuesto de new, ya que devuelve la memoria asignada por new al "pozo dinámico" y el valor del puntero se vuelve indefinido. El programa DosMasDos lo ilustra:

```
PROGRAM DosMasDos (output);

TYPE
  puntero         = ↑integer;

VAR
    p1,
    p2            : puntero;
    respuesta     : integer;

BEGIN
  new(p1);
  p1↑:=2;
  new(p2);
  p2↑:=p1↑;
  respuesta:=p1↑+p2↑;
  WriteLn (p1↑,'+',p2↑,'=',respuesta);
  dispose (p1);
  dispose (p2);
END.
```

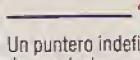
Este método sumamente peculiar de sumar dos más dos demuestra dos puntos importantes:

- Las variables dinámicas son anónimas: no hay ningún identificador de variable (como N) que retenga el valor 2 en ningún lugar del programa; a estos elementos se alude de forma indirecta a través de los punteros.
  Tras el segundo dispose la única memoria que se utiliza es el único entero (respuesta); todos los datos dinámicos ya han dejado de existir.

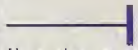
Si usted ha seguido nuestra serie dedicada al lenguaje assembly, ya habrá observado que la indirección implícita en el uso de punteros es análoga al direccionamiento indirecto a nivel de lenguaje máquina. Sin embargo, hay grandes diferencias en el uso de la indirección en un lenguaje de tan alto nivel como el PASCAL. En primer lugar, nunca sabemos (o no necesitamos saber) cuáles son las verdaderas direcciones. La única "dirección absoluta" de que dispone el programador de PASCAL es NIL.

- Asimismo, somos libres de utilizar, reclamar y posteriormente reutilizar la memoria disponible sin necesidad de organizar nosotros mismos ninguna "recolección de basura". Es el PASCAL el que se ocupa de la gestión de la memoria, y la única información que quizá podríamos necesitar es cuánta

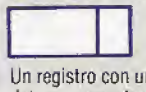
Símbolos de puntero



Un puntero indefinido (después de una declar., antes de una asign. o después de *dispose*)



Un puntero que no señala hacia "ningún sitio" (después de una asign. a NIL)



Un registro con un campo de datos y un puntero

memoria queda. Esto se obtiene mediante la función no estandarizada MemAvail o, en algunas implementaciones, como en el ISO PASCAL Acorn, mediante la función Free. Ésta devuelve la cantidad de bytes de memoria que quedan disponibles.

Otra ampliación útil es la función SizeOf (IdentificadorTipo), que devuelve el tamaño, en bytes, de cualquier tipo determinado. La RAM para el usuario se divide en dos estructuras internas: la pila y el *heap* (montón).

La pila se emplea para llamadas a procedimientos y funciones, almacenando sus direcciones de retorno, datos locales y valores devueltos. Todos los datos dinámicos se asignan al montón. Éste opera de forma similar a la pila, a excepción de que no es una estructura de datos LIFO (último en entrar, primero en salir) y el montón comienza en el extremo opuesto de la memoria para el usuario, creciendo en dirección a la pila. El uso excesivamente ambicioso de new y/o la recursión podría producir una "colisión pila-montón", pero ésta se puede evitar mediante:

```
IF SizeOf (cosa)>PerCent*MemAvail THEN..
```

donde cosa es el identificador de tipo de los datos que están por crear y PerCent es un valor del orden de 0.7, permitiendo el 30 % de la memoria para la pila y un 70 % para el montón. Si, como suele ser el caso, MemAvail/Free opera como una "línea de marea alta", se podría mantener un "contador de basura" adicional de elementos desechados.

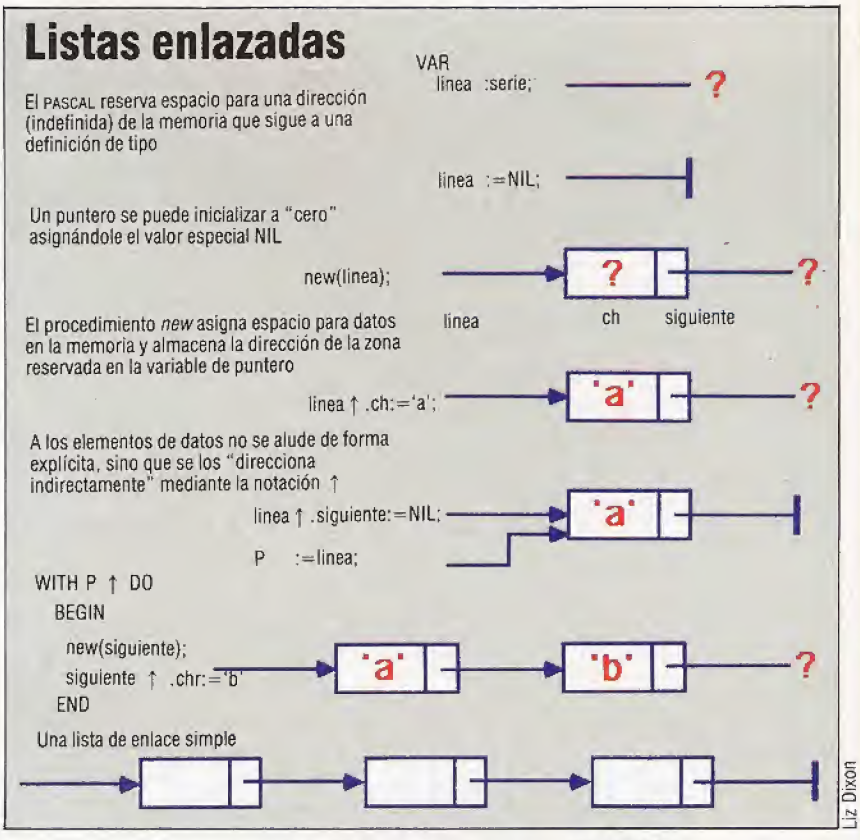
Estructuras enlazadas

El verdadero poder de los punteros queda de manifiesto cuando creamos estructuras enlazadas tales como árboles, listas de enlace simple o doble, estructuras circulares, etc. Si consideramos el problema de series de caracteres, podemos usar, y a menudo lo hacemos, una matriz de algún tamaño estipulado, pongamos 80 caracteres. Si tenemos una matriz de series para almacenar un documento, por ejemplo, cada línea en blanco consumirá aún 80 bytes de almacenamiento. Igualmente, no podemos representar líneas de longitud superior a 80 caracteres: toda la estructura de datos es, sencillamente, demasiado rígida.

Un compilador de PASCAL ha de distinguir identificadores de cualquier longitud. Entonces, ¿cómo podemos, por ejemplo, reproducir con precisión esta característica del mundo real de tamaño variable? La estructura natural a utilizar sería un registro que contuviera dos campos: uno para cada elemento de datos (chars en este caso) y otro de puntero que señalara hacia el registro siguiente, si lo hubiera, de la lista.

```
TYPE
     serie      = ↑ caracter;
     caracter   =RECORD
                   ch :char;
                   siguiente :serie
                   END;

VAR
     linea      :serie;
BEGIN
  linea:=NIL;
        etc.
```

Listas enlazadas



Liz Dixon

Una serie vacía se representa asignándole a la serie el valor NIL. Cualquier otra secuencia de caracteres exigirá un nuevo registro que contenga cada char y otro puntero que apunte al siguiente registro. El último registro tendrá su campo siguiente inicializado en NIL, de modo que se pueda detectar el final de la serie. Un procedimiento para imprimir la serie sería, entonces:

```
WHILE linea<>NIL DO
  BEGIN
    write (linea ↑ .ch);
    linea:=linea ↑ .siguiente
  END
```

Observe que la estructura de datos es recursiva, puesto que se define en términos de sí misma. Al campo de puntero no se le puede dar un tipo que se haya definido totalmente, de modo que se permite una referencia "hacia adelante"

En sucesión
El PASCAL ofrece al programador la posibilidad de implementar "listas enlazadas", poderosas estructuras de datos en las que cada elemento apunta al siguiente elemento de la lista. Las listas pueden ser de enlace simple (entre elementos sucesivos), de enlace doble (cada elemento posee punteros tanto hacia el elemento siguiente como hacia el elemento anterior de la lista), o circulares (donde el último elemento de una lista de enlace simple señala hacia atrás, hacia el primer elemento de la lista)

Programa de lista circular

```
PROGRAM   ListaCirc   (input, output);
{
——
—— Intención: Insertar registros que contienen
—— datos mezclados en una lista circular
—— asignada dinamicamente. Los datos se colocan POR orden
—— DE un campo alfabetico clave (Nombre) y, por consiguiente,
—— son innecesarios algoritmos de clasificacion.
——

CONST
        LongitudSerie   =25;
        espacio         =' ';

TYPE
        Cardinal        = 0..MaxInt;
        TamañoSerie     = 1..LongitudSerie;
        serie           = PACKED ARRAY [TamañoSerie]
                            OF char;
        cosa            = RECORD
                            Nombre      :serie;
                            {..otros campos}
                            deuda       :Cardinal
                          END; {cosa}
        puntero         = ↑ nudo;
```

La línea circular
Este programa le permite insertar registros que contengan datos mezclados en una lista circular asignada dinámicamente. Los datos se colocan por orden de un campo clave alfabético (Nombre) y, por consiguiente, no se requieren algoritmos de clasificación

```
        nudo            = RECORD
                            item        : cosa;
                            siguiente   : puntero
                          END;          {nudo}
VAR
    lista               : puntero;

{1111111111111111111111111111111111111111}

PROCEDURE  LeerCosas                    ( encabezamiento: puntero);
  VAR
        datos       : cosa;
        IK          : boolean;

  {22222222222222222222222222222222}

  PROCEDURE  SaltoarBlancos         ( VAR  F  : text );
    {ignorar espacio en blanco de adelante}
    VAR
        hecho       : boolean;
    BEGIN
      hecho := EoF (F);

      WHILE NOT hecho DO
        BEGIN
          hecho := (F ↑ >espacio) OR EoLn (F);

          IF NOT hecho THEN
            BEGIN
              get (F);
              hecho := EoF (F)
            END
      END;  {SaltarBlancos}
    {22222222222222222222222222222222}

  PROCEDURE  LeerFicha                  ( VAR   F   : text;
                                          VAR   N   : Cardinal;
                                          VAR   OK  : boolean);
    {recoger un numero no negativo}
    TYPE
        simple      = 0..9;
    VAR
        digitos     : SET OF char;
        hecho       : boolean;

        {33333333333333333333}

    FUNCTION Valor (digito  :char)      : simple;
      {convertir char A valor numerico}
      BEGIN
        Valor       := ord (digito) – ord ('0')

      END;          {Valor}
          {33333333333333333333}

    FUNCTION Legal          ( N         : Cardinal;
                              digito    : char)         : boolean;
      {estamos POR superar MaxInt ?}
      BEGIN
        IF N = MaxInt DIV 10
          THEN
            Legal := Valor (digito)<=MaxInt MOD 10
          ELSE
            Legal := N<MaxInt DIV 10

      END;          {Legal}
          {33333333333322222222222}
    BEGIN       {LeerFicha}

      digitos := ['0' ..'9'];
      OK := NOT EoF (F);

      IF OK THEN {¿hallado un digito?}
        OK := F ↑ IN digitos;

      N := 0;
      hecho := NOT OK;

      WHILE OK AND NOT hecho DO
        BEGIN {llevar N A base 10:}
          N := 10*N+Valor (F ↑ );
          get (F);
          hecho := EoF (F);

          IF NOT hecho THEN
            hecho := NOT (F ↑ IN digitos);

          IF NOT hecho THEN
            OK := Legal (N,F ↑ )
        END;

      IF NOT EoF (F) THEN
        ReadLn (F)

    END;        {LeerFicha}
    {22222222222222222222222222222222}
  PROCEDURE  LeerLinea ( VAR S  : serie );
    {leer una linea DE chards desde el teclado}
    VAR

        indice      : 0..LongitudSerie;
        simbolo     : chard;
    BEGIN
      SaltarBlancos (input);
      indice :=0;

      WHILE NOT EoLn AND (indice<Longitud Serie) DO
        BEGIN {colocar un char en la serie}
          indice := succ (indice);
          read (simbolo);
          S [indice]:=simbolo
        END;

      IF indice<LongitudSerie
        THEN {rellenar con NULos}
          FOR indice := indice+1 TO LongitudSerie DO
            S [indice] := chr (0)
        ELSE
          IF NOT EoLn THEN {demasiados chars}
            WriteLn ('ATENCION– truncada la entrada');

      ReadLn

    END;        {LeerLinea}
      {22222222222222222222222222222222}

  PROCEDURE  Insertar           ( lista             : puntero;
                                  datos             : cosa);
    VAR
        END

        explorador,
        enlace          : puntero;
        alpha           : serie;
    BEGIN {colocar datos en el nudo de cabeza:}
      lista ↑ item := datos;
      explorador := lista;
      alpha := explorador ↑ .siguiente.item.Nombre;

      WHILE datos .Nombre<alpha DO
        BEGIN {recorrer la lista}
          explorador := explorador ↑ .siguiente;
          alpha := explorador ↑ .siguiente ↑ .item.Nombre
        END;
            {ninguna comprobación DE duplicados}
      new (enlace);             {crear otro nudo}
      enlace ↑ .item := datos; {insertar los datos,}
      enlace ↑ .siguiente:=explorador ↑ .siguiente; {& enlace}
      explorador ↑ .siguiente:=enlace       {en la lista}

    END;        {Insertar}
        {22222222222222222211111111111111111}

BEGIN       {LeerCosas}

  write     ('Nombre?' );
      LeerLinea (datos.Nombre);
                {detenerse en una entrada nula:}
      WHILE datos . Nombre [1] <> chr (0) DO
        BEGIN
          REPEAT
            REPEAT
              write ('Cantidad adeudada ?':20);

            IF EoLn (input) THEN
              ReadLn (input);

            SaltarBlancos (input)

          UNTIL NOT EoLn (input);

          LeerFicha (input, datos. deuda. OK);

          IF NOT OK THEN [ningun digito]
            WriteLn ('***ERROR–':20,
                     'por favor vuelva a entrar***')

        UNTIL OK;

        Insertar (encabezamiento, datos);
        WriteLn ('(RETURN al acabar)' :40);
        write ('Nombre?');
        LeerLinea (datos. Nombre)
      END

  END;      {LeerCosas}
  {11111111111111111111111111111111111111}

PROCEDURE Visualizar (lista   :puntero);
  VAR

      imprimir      : puntero;
  BEGIN       {primero saltar el nudo de cabeza ficticio:}
    imprimir := lista ↑ .siguiente;
    page (output);
    WriteLn ('Lista ordenada:');
    WriteLn;
            {algun dato (mas) A imprimir?}
    WHILE NOT (imprimir=lista) DO
      BEGIN
        WITH imprimir ↑ .item DO
          WriteLn (deuda : 8, Nombre : 30);

        WriteLn;
        imprimir := imprimir ↑ .siguiente
      END

  END;      {Visualizacion}
  {11111111111111111111111111111111111111}

BEGIN       {ListaCirc– PROGRAMA principal}

  new (lista);      {crear un nudo de cabeza ficticio}
  lista ↑ .siguiente := lista; {hacer que se apunte A sí mismo}
  page (output);
  WriteLn ('Entrar datos :   primero el apellido, luego');
  WriteLn ('la cantidad adeudada en ptas. (sin decimales) :');
  WriteLn;
  Leer Cosas (lista);                   {poner datos en la lista}
  Visualizar (lista);                   {ahora imprimirla POR orden}
  WriteLn ('---hecho---' : 40)

END.
```

Tecnomúsica

El SID (Sound Interface Device: dispositivo interface para sonido) es el responsable de las posibilidades sonoras del Commodore 64

El sonido emitido por el Commodore 64 suele canalizarse a través del enchufe RF que conecta directamente con el televisor. Pero la salida sonora puede igualmente ser dirigida a través de un enchufe de audio/video, a un sistema *hi-fi* para su reproducción o grabación perfecta. En este capítulo estudiaremos los principios subyacentes en el software que va a convertir al Commodore 64 en una caja de ritmos.

La calidad del resultado final se mejora sensiblemente si la reproducción se realiza por medio de un sistema de alta fidelidad.

Además de crear sofisticados sonidos bajo control de software, el chip SID puede aceptar señales acústicas externas. Tales señales pueden ser generadas por hardware externo electrónico (que quizá incorpore otro chip SID) o por instrumentos musicales tales como la guitarra eléctrica. Esta señal externa puede mezclarse con la salida acústica del chip SID y ser procesada por sus filtros. Sin embargo, debemos aconsejar cautela en el empleo de las interfaces de E/S, ya que una conexión incorrecta de las líneas externas puede dañar seriamente al ordenador.

Le sugerimos que consulte el manual del fabricante sobre las salidas correctas y los niveles antes de realizar ninguna conexión externa.

Para comenzar el análisis, debemos enfocar nuestra atención en cómo se obtiene un sonido musical (periódico) a partir de notas puras individuales. Consideraremos después otro método alternativo para conseguir el mismo resultado. Por medio del control de la envoltura de un determinado sonido periódico, podemos introducir armónicos en diversas proporciones. Esto en la práctica significa que podemos crear casi cualquier sonido periódico que deseemos mediante una sencilla alteración de unos pocos "ingredientes" de la envoltura de un diapasón. Estos ingredientes —o puntos de control— son conocidos por las siglas inglesas ADSR (*attack, decay, sustain, release*: subida, bajada, retención, final).

Por desgracia el BASIC del Commodore 64 no proporciona muchas facilidades para el tratamiento de los sonidos. La programación sonora se basa fundamentalmente en las instrucciones PEEK y POKE. Dando a la variable SID el valor 54272 (la dirección de base del chip SID) hacemos que las direcciones SID y siguientes hasta la SID+28 controlen el chip del sonido y, por tanto, toda la capacidad sonora del Commodore 64.

Por último, proporcionaremos un programa en código máquina que creará una máquina tambor o de ritmos gestionada por interrupciones y a tres voces. Esta cuña en código máquina puede ser controlada desde el BASIC, pero las percusiones continuarán con independencia del programa en BASIC. Lo que significa que, con ligeras modificaciones, podemos proporcionar sonidos de fondo a cualquier programa en BASIC.

El sonido llega a nuestros oídos en forma de vibraciones periódicas del aire. El número de vibraciones por segundo se denomina *altura* (o *frecuencia*) de un sonido. El umbral inferior de sensibilidad del oído humano está cifrado en 15 ciclos por segundo (15 hertzios). Una nota pura de 100 Hz nos parece baja. La nota *la* inmediatamente superior al *do* normal o central tiene 440 Hz como cifra universalmente aceptada. Si se dobla la frecuencia de una nota, elevamos su altura a una octava justa. Un oído humano normal puede percibir hasta 10,5 octavas. Los osciladores de tres notas del chip SID sólo se mueven en un arco de ocho octavas (aproximadamente desde 0 Hz hasta 4 000 Hz).

Jean Fourier (1768-1830), físico francés, fue el primero que observó que toda forma de onda periódica puede descomponerse en una nota pura fundamental mezclada con otras notas cuyas frecuencias son múltiplos de dicha nota fundamental. Son los conocidos *armónicos*. El timbre especial que distingue una nota de otra de igual frecuencia es precisamente obra de los armónicos que la acompañan.

Una onda senoidal pura, que corresponderá a una nota pura, es en esencia una señal analógica, lo que quiere decir que no es tan fácil de producir en un dispositivo digital con sólo dos niveles de voltaje, 0 V o 5 V. Por ello, en lugar de generar frecuencias puras y mezclarlas después para obtener el sonido periódico deseado, debemos adoptar otro método con un ordenador personal.

El chip SID está equipado con tres voces, cada una de las cuales puede generar una de las cuatro ondas periódicas distintas, que son:

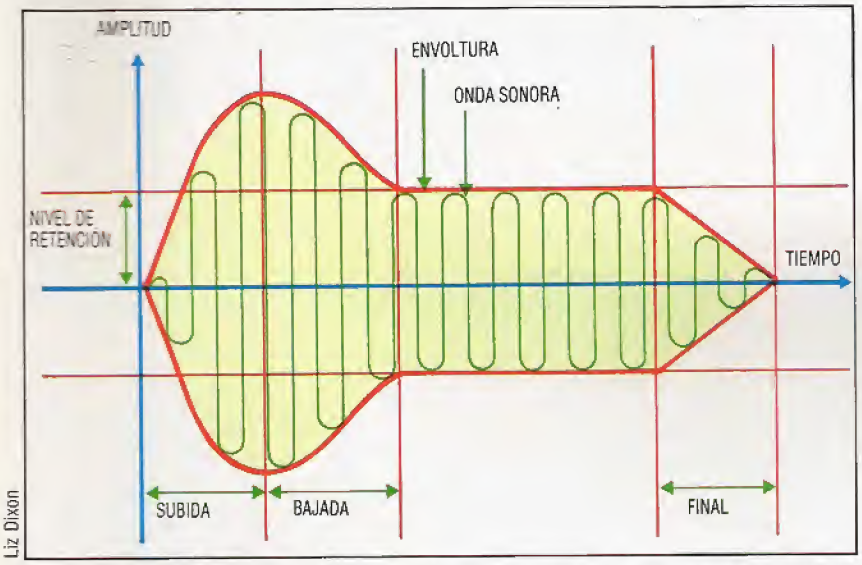
1) *Onda sierra:* contiene todos los armónicos. El armónico enésimo tiene una intensidad proporcional a 1/N.
2) *Onda triangular:* sólo contiene los armónicos impares. Para un N impar, el armónico enésimo tiene una intensidad proporcional a $1/N^2$.
3) *Onda rectangular:* es una onda cuadrada que tiene armónicos pares proporcionales a 1/N. Cambiando la "amplitud de pulsación" se puede obtener una extensa gama de ondas rectangulares, cada una de las cuales con su propia mezcla de armónicos.
4) *Ruido blanco:* es una mezcla aleatoria de frecuencias que se emplea sobre todo para efectos especiales.

El contenido armónico de un sonido también puede alterarse filtrándolo. El chip SID presenta tres tipos de filtro: paso inferior, paso de banda y paso superior. Así, un filtro de paso inferior dará paso a todas las frecuencias que estén por debajo de un valor determinado y atenuará todas las fre-

cuencias por encima de éste. Con estas facilidades y el control de la envoltura ADSR pueden producirse casi todos los sonidos.

Los programas generadores de habla humana pueden realizarse de varias maneras. Aquí describiremos un método que promete una calidad razonable y un vocabulario ilimitado. En este método, las unidades fundamentales del habla —los fonemas— se codifican en una tabla de valores ADSR. En inglés (para cuyo idioma estamos pensando este método) se conocen 52 fonemas diferentes; por ello, su codificación no presenta mayores problemas. Posteriormente emplearemos un programa en código máquina para traducir el texto en ASC (o sea, el ASCII del Commodore) a una corriente de códigos de fonemas, que se enviará después al chip SID empleando la tabla ADSR. Esto no resulta tan fácil como se explica, puesto que las reglas de traducción de un texto en fonemas son muy complejas. La calidad de los sonidos finales dependerá de esta parte del programa precisamente. El esquema es perfectamente aplicable al Commodore 64 y se comercializan diferentes productos que utilizan esta técnica.

Control de la envoltura



Liz Dixon

Envolturas finísimas
La calidad de una nota (el conjunto de características que nos permiten discernir entre una nota de piano y otra de violín, p. ej.) depende del diseño de la envoltura. En la sintetización electrónica de los sonidos la envoltura se considera integrada por cuatro fases distintas. Son conocidas por subida, bajada, retención y final o por las siglas inglesas ADSR. La longitud de cada fase de la envoltura ADSR puede ser definida con valores colocados (POKE) en los registros del SID. Esto es lo que nos permite sintetizar los sonidos de diferentes instrumentos en el Commodore 64

El dibujo de ADSR muestra la forma general de una nota musical subrayando aquellos aspectos que el chip SID puede controlar. Los cuatro factores de la ADSR son:

1) *Subida (attack):* tiempo de elevación de una nota.
2) *Bajada (decay):* tiempo de bajada de una nota a un nivel estable.
3) *Retención (sustain):* volumen de un nivel estable.
4) *Final (release):* tiempo empleado en bajar hasta cero el volumen de la nota.

Las tres primeras zonas de una nota se controlan una a una por cuartetos o *nybbles* (cuatro bits) en los registros del SID. Lo que significa que cada uno de estos parámetros toma valores entre 0 y 15. La relación entre los valores que han de ser colocados (POKE) en los registros del SID y la temporización real está ilustrada en el siguiente cuadro:

Valor	Subida (tiempo/ciclo)	Relación bajada/final (tiempo/ciclo)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 seg
11	800 ms	2.4 seg
12	1 seg	3 seg
13	3 seg	9 seg
14	5 seg	15 seg
15	8 seg	24 seg

La longitud de la zona de retención se calcula mediante un bucle de retardo. Con el cuadro anterior, los valores ADSR para un sonido de violín serán:

ADSR	Tiempo	Valor en POKE
A	500 ms	10
D	300 ms	8
S	—	—
R	750 ms	9

La obtención de un sonido en el Commodore 64 ha de incluir como mínimo los siguientes pasos. Así:

- Paso 1: dar volumen mediante:

```
POKE SID+24,15
```

- Paso 2: seleccionar la ADSR. Por ejemplo:

```
POKE SID+5,9 :REM VOZ#1, SUBIDA/BAJADA
POKE SID+6,0 :REM VOZ#1, RETENCION/FINAL
```

- Paso 3: seleccionar la frecuencia de cada oscilador. Por ejemplo:

```
POKE SID+1,25 : REM VOZ#1,BYTE SUP. DE FRECUENCIA
POKE SID,0 : REM VOZ#1,BYTE INF. DE FRECUENCIA
```

- Paso 4: seleccionar el tipo deseado de forma de onda. Por ejemplo:

```
POKE SID+4,33 : REM VOZ#1,ONDA SIERRA
```

En este paso se empieza a emitir el sonido (lo que se denomina abrir la "puerta").

- Paso 5: bucle de retardo mientras se emite el sonido en el nivel de retención.
- Paso 6: finalizar la forma de onda. Por ejemplo:

```
POKE SID+4,32 : REM FINAL ONDA SIERRA
```

El procedimiento más sencillo de programar una melodía en un Commodore 64 consiste en establecer los valores iniciales de la ADSR y construir un bucle FOR...NEXT que lea (READ) los datos (DATA) de los bytes *hi/lo* de la frecuencia. Si en DATA incluimos ceros, podemos variar el ritmo de las diferentes voces sin cambiar el retardo.

Caja de ritmos

El programa emplea una cuña IRQ, de modo que, una vez establecidas, las percusiones seguirán sonando con independencia de lo que se esté realizando en BASIC. Tiene además otro aspecto interesante en el modo como comprueba desde el código máquina cuál ha sido la tecla pulsada, inspeccionando el contenido de la posición 197 ($00C5). Se trata de una posición de la página cero que contiene un valor que indica la última tecla pulsada. Un programa en código máquina le puede evitar el problema de una llamada núcleo para obtener la pulsación de una tecla, siempre que $00C5 sea empleado con la suficiente frecuencia.

Nótese que a causa del driver del BASIC, que no deja libre el buffer del teclado por medio de GET o INPUT, el programa debe poner a cero el puntero contenido en la posición 198 ($00C6) en el momento de la salida. Esta posición es de la página cero, que contiene generalmente el número de teclas pulsadas almacenado en el buffer de teclado.

Al utilizar el programa, cada tambor dispone de 16 "trozos de tiempo" en los que puede sonar, y se visualizan en una cuadrícula en pantalla. La elección del 16 facilita la obtención del ritmo rock en cuatro por cuatro. El cursor de sprites se mueve a una de las celdillas de la cuadrícula y en el momento de situarse en la posición correcta se activa/desactiva un "trozo de tiempo" con la tecla Return obteniéndose el ritmo mediante la tecla f1. El menú sobre la pantalla indica las opciones disponibles. Dado que los sonidos se controlan por interrupciones, se puede editar en pantalla el modelo de ritmo mientras éste se ejecuta.

El programa en BASIC incluye todo el código máquina necesario para ejecutar el programa con sentencias DATA, y puede ser entrado y ejecutado tal como está. Damos también un listado del código fuente. Si desea digitarlo y ensamblarlo, puede omitir las líneas que van de 1520 a 1540 y las sentencias DATA entre 1620 y 1940.

En medio de la ejecución, intente obtener el modelo que le presentamos de un ritmo clásico:

Programa BASIC generador de ritmos

```
1000 REM ** GENERADOR DE RITMOS **
1010 PRINTCHR$(147):REM LIMPIA PANTALLA
1020 GOSUB1510:REM ESTABLECE C/M+SPRITE
1030 GOSUB1270:REM ESTABLECE PANTALLA
1040 SYS(49152):REM "INSERTA CUÑA"
1050 REM SYS (49175) PARA QUITAR CUÑA
1060 REM "BUCLE PRINCIPAL"
1070 P=PEEK(197):REM TECLA PULSADA
1080 IFP=37THENY=Y-16:SY=SY-1:IFY<95THENY=95:SY=0
1090 IFP=36THENY=Y+16:SY=SY+1:IFY>127THENY=127:SY=2
1100 IFP=47THENX=X-16:SX=SX-1:IFX<28THENX=28:SX=0
1110 IFP=44THENX=X+16:SX=SX+1:IFX>268THENX=268:SX=15
1120 Z=0:IFP=1 THENZ=1
1130 PRINTCHR$(19);S$;"TEMPO=";PEEK(679);CHR$(157);"  "
1140 POKEVIC+16,X/256:POKEVIC,XAND255
1150 POKEVIC+1,Y
1160 REM**CALCULO SITUACION PANTALLA POR LA POS X,Y DEL SPRITE**
1170 SC=B+INT((Y-50)/8+1)*40+INT((X-18)/8)
1180 IFSY=0THENROW=50000:REM RITMO DE BAJO
1190 IFSY=1THENROW=50016:REM RITMO DE TIMBAL
1200 IFSY=2THENROW=50032:REM CENCERRO
1210 IFZ=1ANDPEEK(SC)=32THENPOKESC,24:POKEROW+SX,1:GOTO1230
1220 IFZ=1ANDPEEK(SC)=24THENPOKESC,32:POKEROW+SX,0
1230 IFP=51THENGOSUB1270:FORI=0TO50:POKE50000+I,0:NEXT
1240 IFP=57THENPOKE198,0:POKEVIC+21,0:PRINTCHR$(147):END
1250 GOTO1070
1260 REM "ESTABLECE PANTALLA"
1270 PRINTCHR$(19);
1280 S$=CHR$(17)+CHR$(17)+CHR$(17)
1290 PRINTTAB(3)"  "
1300 PRINTCHR$(18)TAB(3)" **** GENERADOR RITMOS COMM64 ****"
1310 PRINT:PRINT:PRINT
1320 PRINT" ----------------------------------------
1330 PRINT"                                        -BAJO"
1340 PRINT" ----------------------------------------
1350 PRINT"                                        -TIMBAL"
     "
1360 PRINT" ----------------------------------------
1370 PRINT"                                        -CENCERRO"
1380 PRINT" ----------------------------------------
1390 PRINT:PRINTTAB(8)          "OPCIONES...."
1400 PRINTTAB(8)"[F1]           DAR RITMO"
1410 PRINTTAB(8)"[F3]           REDUCIR TEMPO"
1420 PRINTTAB(8)"[F5]           AUMENTAR TEMPO"
1430 PRINTTAB(8)"[F7]           PARAR RITMO"
1440 PRINTTAB(8)"[CLR]          BORRAR DIAGRAMA"
1450 PRINTTAB(8)"[8T]           FINALIZAR PROGRAMA"
1460 PRINTTAB(8)"[<]            MOVER CURSOR IZQ"
1470 PRINTTAB(8)"[>]            MOVER CURSOR DERECHA"
1480 PRINTTAB(8)"[K]            SUBIR CURSOR"
1490 PRINTTAB(8)"[M]            BAJAR CURSOR"
1500 RETURN
1510 REM "ESTABLECE COD MAQ Y SPRITE"
1520 FORI=49152TO49413
1530 READJ:C=C+J:POKEI,J:NEXTI
1540 READJ:IFC<>JTHENPRINT"ERROR DATOS":END
1550 FORI=0TO62:READJ:POKE832+I,J:NEXTI
1560 VIC=53248:X=28:Y=95:B=1024:SID=54272
1570 FORI=0TO24:POKESID+I,0:NEXTI
1580 POKESID+24,15:POKEVIC+21,1:POKE2040,13
1590 POKE254,15:POKE679,15:POKEVIC+39,1
1600 FORI=0TO50:POKE50000+I,0:NEXT
1610 RETURN
1620 REM***DATOS M/C***
1630 DATA120,173,20,3,133,251,173,21,3
1640 DATA133,252,169,36,141,20,3,169
1650 DATA192,141,21,3,88,96,120,165,251
1660 DATA141,20,3,165,252,141,21,3,88
1670 DATA96,32,177,192,165,197,201,3
1680 DATA208,3,32,135,192,165,197,201,4
1690 DATA208,3,32,140,192,165,197,201,5
1700 DATA208,3,32,155,192,165,197,201,6
1710 DATA208,3,32,166,192,32,87,192,142
1720 DATA168,2,140,169,2,76,49,234,165
1730 DATA253,208,1,96,198,254,240,1,96
1740 DATA32,149,192,224,16,208,3,32,144
1750 DATA192,185,80,195,240,3,32,184
1760 DATA192,185,96,195,240,3,32,210
1770 DATA192,185,112,195,240,3,32,236
1780 DATA192,200,234,232,96,169,0,133
1790 DATA253,96,169,1,133,253,162,0,160
1800 DATA0,96,173,167,2,133,254,96,173
1810 DATA167,2,201,255,240,3,238,167,2
1820 DATA96,173,167,2,201,1,240,3,206
1830 DATA167,2,96,174,168,2,172,169,2
1840 DATA96,169,14,141,6,212,169,32,141
1850 DATA2,212,169,66,141,4,212,169,3
1860 DATA141,1,212,169,65,141,4,212,96
1870 DATA169,7,141,12,212,169,12,141,13
1880 DATA212,169,128,141,11,212,169,65
1890 DATA141,8,212,169,129,141,11,212
1900 DATA96,169,2,141,19,212,169,13,141
1910 DATA20,212,169,18,141,18,212,169
1920 DATA100,141,15,212,169,17,141,18
1930 DATA212,96
1940 DATA32038:REM "SUMA DE CONTROL"
1950 REM**DATOS CURSOR SPRITE**
1960 DATA127,254,0,127,254,0,127,254,0
1970 DATA112,14,0,112,14,0,112,14,0,112
1980 DATA14,0,112,14,0,112,14,0,112,14
1990 DATA0,112,14,0,127,254,0,127,254,0
2000 DATA127,254,0,0,0,0,0,0,0,0,0,0,0,0
2010 DATA,0,0,0,0,0,0,0,0,0,0,0
```

Listado assembly

```
; ++++++++++++++++++
; ++++++++++++++++++
; ++                ++
; ++   CODIGO FUENTE   ++
; ++  CAJA RITMOS CBM  ++
; ++                ++
; ++++++++++++++++++
; ++++++++++++++++++
;
VOL     =   $D418     ;VOLUMEN DEL SID
ATT1    =   $D405     ;SUBIDA VOZ 1
SUS1    =   $D406     ;RETENCION VOZ 1
PULSE   =   $D402     ;VELOCIDAD PULSACIONES VOZ 1
WAVE1   =   $D404     ;FORMA ONDA VOZ 1
BASS    =   $D401     ;FRECUENCIA BYTE HI VOZ 1
ATT2    =   $D40C     ;SUBIDA VOZ 2
SUS2    =   $D40D     ;RETENCION VOZ 2
WAVE2   =   $D40B     ;FORMA ONDA VOZ 2
SNARE   =   $D408     ;FRECUENCIA BYTE HI VOZ 2
ATT3    =   $D413     ;SUBIDA VOZ 3
SUS3    =   $D414     ;RETENCION VOZ 3
WAVE3   =   $D412     ;FORMA ONDA VOZ 3
BELL    =   $D40F     ;FRECUENCIA BYTE HI VOZ 3
ROW1    =   $C350     ;ALMACENAMIENTO VOZ 1
ROW2    =   $C360     ;ALMACENAMIENTO VOZ 2
ROW3    =   $C370     ;ALMACENAMIENTO VOZ 3
TEMPO   =   $02A7     ;ALMACENAM. TEMPORAL DEL RETARDO
XCOUNT  =   $02A8     ;ALMACENAM. TEMP. DEL REGISTRO X
YCOUNT  =   $02A9     ;ALMACENAMIENTO TEMP. DEL REG Y
LOVEC   =   $FB       ;ALMACENAM. VECTOR BYTES LO
HIVEC   =   $FC       ;ALMACENAM VECTOR BYTES HI
PLAY    =   $FD       ;SONIDO TAMBOR (1=SI)
DELAY   =   $FE       ;ALMACENAM. ESTADO ACTUAL RETARDO
KEY     =   $C5       ;TECLA PULSADA
;-------------------------------------------------
*       =   $C000     ;ASSEMBLE DESDE LA 49152 (DECIMAL)
;-------------------------------------------------
; ESTABLECER CUÑA
;-------------------------------------------------
        SEI           ;DESACTIVA PETICION INTERRUPCIONES
        LDA  $0314    ;TOMA VALORES DEL BYTE LO DEL VECTOR
        STA  LOVEC    ;LOS ALMACENA EN LOVEC
        LDA  $0315    ;TOMA VALORES DEL BYTE HI DEL VECTOR
        STA  HIVEC    ;LOS ALMACENA EN HIVEC
        LDA  #<WEDGE  ;TOMA BYTE LO DE DIR INICIO CUÑA
        STA  $0314    ;LO ALMACENA EN BYTE LO DE VECTOR IRQ
        LDA  #>WEDGE  ;TOMA BYTE HI DE DIR INICIO CUÑA
        STA  $0315    ;LO ALMACENA EN BYTE HI DE VECTOR IRQ
        CLI           ;REANUDA PETICION INTERRUPCION
        RTS           ;RETURN
;-------------------------------------------------
; QUITAR CUÑA
;-------------------------------------------------
        SEI           ;DESACTIVA PETICION INTERRUPCIONES
        LDA  LOVEC    ;TOMA VALOR ORIGINAL DE LOVEC
        STA  $0314    ;LO ALMACENA EN BYTE LO DE VECTOR IRQ
        LDA  HIVEC    ;TOMA VALOR ORIGINAL DE HIVEC
        STA  $0315    ;LO ALMACENA EN BYTE HI DE VECTOR IRQ
        CLI           ;REANUDA PETICION INTERRUPCION
        RTS           ;RETURN
;-------------------------------------------------
; BUCLE PRINCIPAL
;-------------------------------------------------
WEDGE   JSR  REG      ;GOSUB REG
        LDA  KEY      ;QUE TECLA FUE PULSADA?
        CMP  #$03     ;FUE LA TECLA FUNCION #1?
        BNE  CONT1    ;SI NO, BIFURCAR
        JSR  FLAG0    ;GOSUB FLAG0
CONT1   LDA  KEY      ;QUE TECLA FUE PULSADA?
        CMP  #$04     ;FUE LA TECLA FUNCION #7?
        BNE  CONT2    ;SI NO, BIFURCAR
        JSR  FLAG1    ;GOSUB FLAG1
CONT2   LDA  KEY      ;QUE TECLA FUE PULSADA?
        CMP  #$05     ;FUE LA TECLA FUNCION #3?
        BNE  CONT3    ;SI NO, BIFURCAR
        JSR  ADD      ;GOSUB ADD
CONT3   LDA  KEY      ;QUE TECLA FUE PULSADA?
        CMP  #$06     ;FUE LA TECLA #5?
        BNE  CONT4    ;SI NO, BIFURCAR
        JSR  MINUS    ;GOSUB MINUS
CONT4   JSR  REST     ;GOSUB REST
        STX  XCOUNT   ;ALMACENA VALOR REGISTRO X
        STY  YCOUNT   ;ALMACENA VALOR REGISTRO Y
        JMP  $EA31    ;VUELVE A INTERR
;-------------------------------------------------
; RUTINA PRUEBA
;-------------------------------------------------
REST    LDA  PLAY     ;TOMA EL VALOR DEL TOGGLE
        BNE  BEGIN    ;SI ES 1 BIFURCAR
        RTS           ;RETURN
;-------------------------------------------------
BEGIN   DEC  DELAY    ;DECREMENTA EL RETARDO
        BEQ  START    ;BIFURCAR SI ES CERO
        RTS           ;RETURN
;-------------------------------------------------
START   JSR  COUNT    ;CONTADOR GOSUB
        CPX  #$10     ;FIN DEL BUCLE?
        BNE  CHECK    ;SI NO ES 0, BIFURCAR
        JSR  RESET    ;RESTAURA GOSUB
;-------------------------------------------------
CHECK   LDA  ROW1,Y   ;TOMA VALOR DE DESP. FILA 1 POR Y
        BEQ  NEXT1    ;SI ES CERO, BIFURCAR
        JSR  DRUM1    ;GOSUB DRUM 1
NEXT1   LDA  ROW2,Y   ;TOMA VALOR DE DESP. FILA 2 POR Y
        BEQ  NEXT2    ;SI ES CERO, BIFURCAR
        JSR  DRUM2    ;GOSUB DRUM2
NEXT2   LDA  ROW3,Y   ;TOMA VALOR DE DESP. FILA 3 POR Y
        BEQ  NEXT3    ;SI ES CERO, BIFURCAR
        JSR  DRUM3    ;GOSUB DRUM3
NEXT3   INY           ;INCREMENTA DESPLAZAMIENTO
        INX           ;INCREMENTA CONTADOR BUCLE
        RTS           ;RETURN
;-------------------------------------------------
; SUBRUTINAS
;-------------------------------------------------
FLAG0   LDA  #$00     ;ALMACENA CERO –
        STA  PLAY     ;INTRODUCE PLAY
        RTS           ;RETURN
;-------------------------------------------------
FLAG1   LDA  #$01     ;ALMACENA 1 –
        STA  PLAY     ;INTRODUCE PLAY
RESET   LDX  #$00     ;RESTAURA REG X
        LDY  #$00     ;RESTAURA REG Y
        RTS           ;RETURN
;-------------------------------------------------
COUNT   LDA  TEMPO    ;TOMA VALOR TEMPO
        STA  DELAY    ;ALMACENA RETARDO
        RTS           ;RETURN
;-------------------------------------------------
ADD     LDA  TEMPO    ;TOMA VALOR TEMPO
        CMP  #$FF     ;COMPARA RESULTADO CON 255
        BEQ  CONT5    ;SI NO ES 255, BIFURCAR
        INC  TEMPO    ;INCREMENTA TEMPO
CONT5   RTS           ;RETURN
;-------------------------------------------------
MINUS   LDA  TEMPO    ;TOMA VALOR TEMPO
        CMP  #$01     ;COMPARA RESULTADO CON 1
        BEQ  CONT6    ;SI NO ES 1, BIFURCAR
        DEC  TEMPO    ;DECREMENTA TEMPO
CONT6   RTS           ;RETURN
;-------------------------------------------------
REG     LDX  XCOUNT   ;ALMACENA VALOR CONT X EN REG X
        LDY  YCOUNT   ;ALMACENA VALOR CONT Y EN REG Y
        RTS           ;RETURN
;-------------------------------------------------
; RUTINAS TOQUE DE TAMBOR
;-------------------------------------------------
DRUM1   LDA  #$0E     ;ESTA-
        STA  SUS1     ;
        LDA  #$20     ;BLECER
        STA  PULSE    ;
        LDA  #$42     ;TAMBOR
        STA  WAVE1    ;
        LDA  #$03     ;BAJO
        STA  BASS     ;
        LDA  #$41     ;Y TOCAR
        STA  WAVE1    ;
        RTS           ;VOLVER
;-------------------------------------------------
DRUM2   LDA  #$07     ;ESTA-
        STA  ATT2     ;
        LDA  #$0C     ;BLECER
        STA  SUS2     ;
        LDA  #$80     ;TIMBAL
        STA  WAVE2    ;
        LDA  #$41     ;Y
        STA  SNARE    ;
        LDA  #$81     ;TOCAR
        STA  WAVE2    ;
        RTS           ;VOLVER
;-------------------------------------------------
DRUM3   LDA  #$02     ;ESTA-
        STA  ATT3     ;
        LDA  #$0D     ;BLECER
        STA  SUS3     ;
        LDA  #$12     ;CENCERRO
        STA  WAVE3    ;
        LDA  #$64     ;Y
        STA  BELL     ;
        LDA  #$11     ;TOCAR
        STA  WAVE3    ;
        RTS           ;VOLVER
        .END
```

Steve Cross

La carrera de las ratas
El diagrama ilustra tres rutas posibles a través del laberinto utilizando tres estrategias de búsqueda diferentes: aleatoria, sistemática y heurística. En este ejemplo, el método heurístico encuentra el objetivo antes que los otros dos métodos, si bien sería posible diseñar un laberinto en el cual no fuera así. En la práctica, a menudo los laberintos se resuelven mejor mediante una combinación de métodos de búsqueda heurístico y exhaustivo (en el cual se consideran todas las posibilidades)

Búsqueda inteligente

El desarrollo de técnicas de búsqueda constituye un importante área en la investigación de la inteligencia artificial

Imaginemos que se sueltan tres ratas en un laberinto en cuyo interior hay, en algún lugar, un cuenco lleno de apetitosas bolitas para ratas. Una de ellas deambula por el laberinto durante algunos minutos y después se queda dormida (el insensible experimentador ha estado echando ginebra en el agua de la que bebía el animal). La segunda rata es más metódica: coloca su pata delantera izquierda contra un pared y prosigue efectuando giros a la izquierda ante cada bifurcación, retrocediendo cuando detecta una calle sin salida. Finalmente alcanza su objetivo, pero para entonces la tercera rata ya se ha comido todo el alimento.

La tercera rata posee un delicado sentido del olfato. En varios puntos durante la búsqueda se detiene a olfatear el aire y sigue el camino que, según percibe, la conducirá más cerca del delicioso aroma. Es posible idear un laberinto que despiste a esta rata (así como es posible idear uno que confunda a la seguidora de paredes), pero casi todos estaríamos de acuerdo en que este tipo de estrategia de búsqueda es la más inteligente de las tres.

El concepto de búsqueda es clave en el campo de la inteligencia artificial. Tanto si se halla a 70 m de profundidad en las aguas del Caribe con una escafandra autónoma de submarinista buscando un tesoro sumergido, como sentado a una mesa, concentrado en la resolución de un crucigrama, usted va en busca de algo. El solucionador de problemas de ecuaciones metafóricas mediante búsqueda ha demostrado ser muy provechoso en la AI, porque se pueden tratar muchos problemas diferentes mediante el empleo de técnicas de búsqueda. Nuestras tres ratas ilustran diferentes clases de estrategia:

- Búsqueda aleatoria (el "recorrido de borracho")
- Búsqueda exhaustiva (enumeración sistemática)
- Búsqueda heurística (exploración guiada)

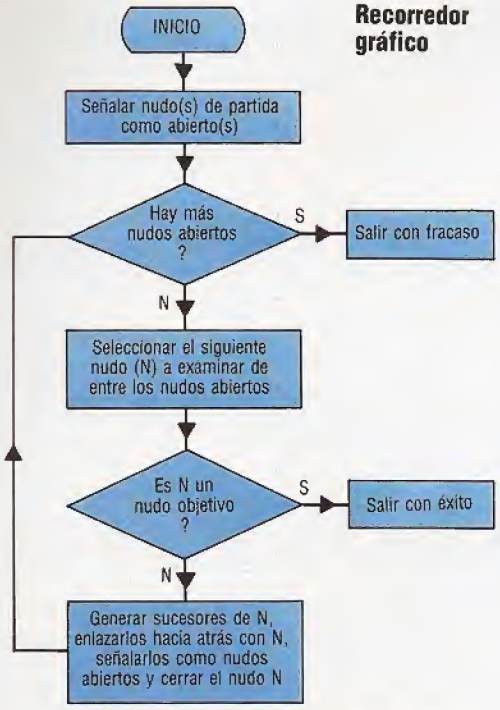
Se considera que el tercero es un enfoque más inteligente al problema que los otros dos, porque normalmente requiere menos esfuerzo para llegar a la solución. Pero todos los métodos heurísticos dependen de alguna forma de saber cuándo la búsqueda se está aproximando a su objetivo (el sentido del olfato de la rata, p. ej.). Sin conocimientos no se puede ser inteligente: uno tiene que depender de la enumeración sistemática.

Hallar una ruta a través de un laberinto es un problema de búsqueda típico. (Sin embargo, es importante comprender que muchos problemas que no son espaciales, como la integración de una expresión matemática simbólica, se pueden tratar uti-

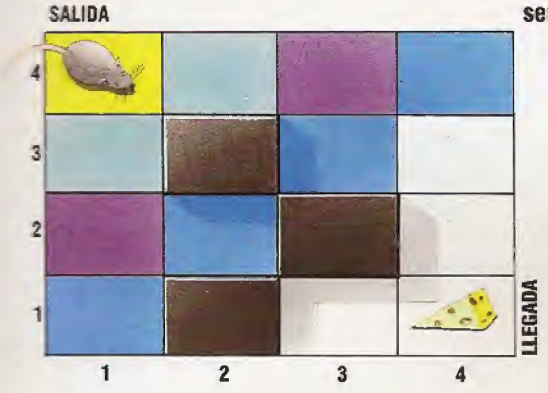
Cuestiones de búsqueda

La mayoría de los métodos de búsqueda de AI implican la generación de árboles a partir del trazado del laberinto a recorrer. El árbol que vemos aquí se configura a partir del laberinto tomando el punto de partida como el nudo raíz, y produciendo sucesivas generaciones mediante la consideración de todos los movimientos posibles desde el nudo en cuestión. Observe que no se permiten movimientos en diagonal. El diagrama de flujo esboza el algoritmo básico para generar la estructura arborescente

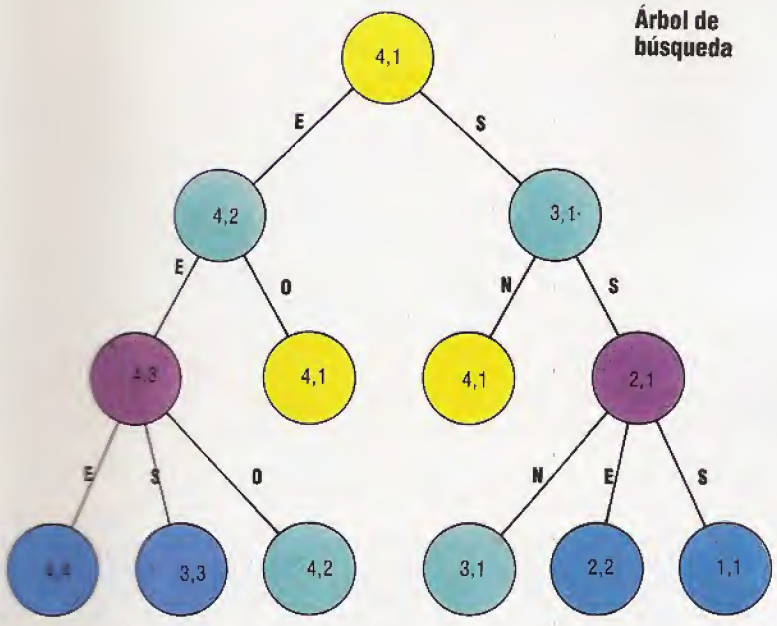
Recorredor gráfico



Un laberinto sencillo



Árbol de búsqueda



Kevin Jones

lizando el mismo marco de búsqueda.) Existen dos formas de juzgar la calidad de un método de búsqueda:

- ¿Cuánto tiempo se necesita para hallar una ruta?
- ¿Es la hallada la ruta más económica?

Lo ideal es contar con un método que encuentre la solución óptima en el menor tiempo posible. En la práctica, no obstante, podemos vernos obligados a optar ya sea por aceptar una solución rápida por debajo del nivel óptimo o bien tomarnos mucho tiempo para la mejor solución posible.

La mayor parte de las estrategias de búsqueda en inteligencia artificial se crean según un plan común, que permite dar mayor o menor énfasis a uno u otro de estos criterios de actuación. El diagrama de algoritmos de recorrido que vemos esboza una familia de métodos de búsqueda a partir de los cuales se puede seleccionar un miembro determinado mediante la elección de cómo rellenar el casillero central, o de cómo decidir qué nudo examinar a continuación. Los términos "abierto" y "cerrado" aluden al estado de cualquier nudo del camino desde el principio al punto de destino. Los nudos abiertos necesitan ser examinados; los nudos cerrados ya se han examinado. Los nudos abiertos se hallan en una especie de lista de espera, y la clave para la eficacia es el orden por el cual se procesará esta lista.

La búsqueda avanza configurando un árbol. El sencillo laberinto de 4×4 (izquierda) muestra a la rata-robot en la casilla superior izquierda y el alimento en la inferior derecha. En cada casilla la rata tiene una opción de hasta cuatro movimientos (norte, sur, este u oeste), si bien algunos pueden estar bloqueados. Si dibujamos las opciones abiertas a la rata al principio, podrá ver la estructura arborescente del proceso de búsqueda.

Desde el cuadrado inicial de (4,1), la rata sólo puede avanzar hasta el este a (4,2) o hacia el sur a (3,1). Desde (4,2) puede avanzar hasta el este o el oeste y desde (3,1) puede ir hacia el norte o hacia el sur. Si el roedor se mueve hacia el este y luego hacia el oeste (o al sur y después al norte) volverá a llegar al punto de partida, lo que no es muy inteligente pero refleja cómo la misma casilla puede aparecer en diferentes nudos del árbol, correspondiendo a las diversas formas de llegar a ellos.

Un método sistemático para explorar el árbol es el llamado *búsqueda por niveles*. Investiga los nudos por orden de proximidad al punto de partida o raíz. Por tanto, considera cada secuencia de N movimientos (a nivel N en el árbol) antes de cualquier secuencia de N+1 movimientos. Está destinado a hallar una ruta que exija la menor cantidad posible de pasos y, dado que cada paso es igualmente costoso, resultará ser la solución de mínimo costo. Pero hallarla podría llevar mucho tiempo. A medida que el árbol se vuelve más ancho y más profundo, el tiempo requerido para hallar el objetivo aumenta de forma exponencial.

Para mejorar una búsqueda por niveles se requiere una fuente de información heurística acerca de la distancia a la que se halla el objetivo. Como dicha medida podríamos utilizar la *Manhattan distance* (distancia de Manhattan). En la isla de Manhattan (estado de Nueva York), la mayoría de las calles se intersectan en ángulo recto. Para llegar de A a B uno debe avanzar tantas manzanas hacia el

norte o el sur y tantas otras hacia el este o el oeste. Del mismo modo, en el laberinto, una rata inteligente puede calcular a cuántas casillas de distancia se halla el objetivo.

Si se sabe que una búsqueda se está aproximando a la solución, puede ser acelerada. La rata de nuestro ejemplo inicial seguía una regla sencilla: acercarse cada vez más al objetivo. Ésta es la estrategia de *escalar montañas*, así llamada porque se la puede comparar con hallar la cima de una montaña cuando uno está desorientado avanzando siempre hacia arriba. Puede ser mucho más rápida que una búsqueda por niveles, pero no ofrece garantías de hallar la ruta óptima. Podría quedarse varada en un pico local (porque la búsqueda está orientada hacia *cualquier* cima).

Si se realiza una síntesis entre la estrategia de búsqueda por niveles y la de escalar montañas surge un método mejor: el llamado *algoritmo A-Star* (A*). Éste selecciona el nudo a examinar a continuación sobre la base de: HD+SF, donde HD es el cálculo heurístico de la distancia que queda y SF es la distancia cubierta hasta el momento. Cuanto mejor sea la estimación de HD, más eficaz es la búsqueda. Si el HD falla, estará evaluado en menos en vez de en más.

En nuestro programa de ejemplo, utilizamos una estructura de programa para implementar estos tres métodos, sólo con modificaciones menores. Las únicas diferencias se hacen evidentes en la elección del nudo a examinar a continuación:

Búsqueda por niveles	— tomar SF más bajo
Escalar montañas	— tomar HD más bajo
Algoritmo A*	— tomar HD+SF más bajo

Programa de búsqueda para el laberinto

```
1000 REM ******************************************
1010 REM ** Listado 2.1:                          **
1020 REM **       PROGRAMA BUSQUEDA LABERINTO     **
1030 REM ******************************************
1040 MODE7
1050 MH=17:MW=25:REM Altura y anchura del laberinto
1060 SI=256 : REM limites del arbol.
1070 WA=1: RR=2 :FO=3 :DN=4 :BL=5
1080 W1=1: REM peso de SF
1090 W2=2: REM peso de HD
1100 DIM M(MH+1,MW+1):REM el laberinto
1110 DIM C$(5) : REM caracteres del laberinto
1120 DIM P(SI),S(SI),N(SI),H(SI)
1130 REM Camino, Pasos,Nudo,Distancia-heuristica.
1140
1150 REM — Rata en el laberinto:
1160 GOSUB 1360 : REM hacer el laberinto
1170 NC=0 : REM n. de nudos examinados
1180 K=0 : REM contador
1190 GOSUB 1660 : REM borrar todos los caminos
1200 N(1)=2*MW+2 : REM 1er. nudo abierto
1210 S(1)=0: H(1)=FR-1+(FC-1)
1220 P(1)=0: REM ningun predecesor
1230 REM — Bucle principal:
1240 REM **** BUCLE PRINCIPAL ****
1250 GOSUB 1770 : REM tomar siguiente nudo S
1260 NC=NC+1
1270 PRINT TAB (0,22);NC,SR,SC,H(S);"
1280 GOSUB 1880 : REM generar sucesores
1290 IF (FR<>SR OR FC<>SC) AND NC<300 THEN 1240
1300 PRINT TAB (0,22);"Concluida la busqueda!"
1310 IF FR=SR AND FC=SC THEN GOSUB 2290 : REM rastrear pasos
1320 IF NC>300 THEN PRINT "Fracaso!"
1330 PRINT NC;" nudos examinados."
1340 END
1350
1360 REM — Rutina para crear laberinto:
1370 FOR R=1 TO MH+1
1380 FOR R=1 TO MW+1
1390 M(R,C)=-(RND(1)<.28 THEN M(P,R)=WA ELSE M(P,R)=BL
1400 IF R=0 OR R=1 THEN M(P,R)=BL
1410 IF R=1 OR R=1 THEN M(P,R)=WA
1420 IF R>MH OR R>MW THEN M(P,R)=WA
1430 NEXT:NEXT
1440 FR=2+INT(RND(1)*(MH-1)):REM fila alimento
1450 FC=4+INT(RND(1)*(MW-3): REM columna alimento
1460 M(FR,FC)=FO
1470 M(2,2)=RR : REM rata robot
1480 C$(BL)=" "
1490 C$(WA)=CHR$255 : REM gotita
1500 C$(DN)=" "
1510 C$(RR)="R"
1520 C$(FO)="*"
1530 GOSUB 1560 : REM visualizarlo
1540 RETURN
1550
1560 REM — Rutina visualizacion del laberinto:
1570 CLS
1580 FOR R=1 TO MH+1
1590 FOR C=1 TO MW+1
1600 PRINT TAB(C,R);C$(M(R,C));
1610 NEXT
1620 PRINT
1630 NEXT
1640 RETURN
1650
1660 REM — Rutina borrar arbol
1670 FOR Q=1 TO SI
1690 P(Q)=0
1700 S(Q)=DD
1710 N(Q)=0
1720 H(Q)=DD
1730 NEXT
1740 NN=2 : REM siguiente nudo libre
1750 RETURN
1760 :
1770 REM — Tomar mejor nudo S:
1780 S=1: BN=DD
1790 FOR I=1 TO SI
1800 V=S(I)*W1+ABS(H(I))*W2
1810 IF V<BN AND H(I)>=0 THEN S=I:BN=V
1820 NEXT
1830 IF S=1 THEN PRINT TAB(0,20);"Explorando...."
1840 SR=INT(N(S)/MW)
1850 SC=N(S)-MW*SR
1860 RETURN
1870 :
1880 REM — Rutina para generar sucesores:
1890 IF H(S)=0 THEN RETURN: REM hecho.
1900 REM —Norte:
1910 Y=SR-1 : X=SC
1920 IF Y>1 THEN GOSUB 2090
1930 REM —Este:
1940 Y=SR : X=SC+1
1950 IF X<= MW THEN GOSUB 2090
1960 REM —Sur:
1970 Y=SR+1 : X=SC
1980 IF Y<= MH THEN GOSUB 2090
1990 REM —Oeste:
2000 Y=SR : X=SC-1
2010 IF X>1 THEN GOSUB 2090
2020 REM — tambien cerrar nudo S:
2030 H(S)=-H(S)
2040 IF H(S)>0 THEN PRINT "Ugh!"
2050 PRINT TAB(SC,SR);" ";
2060 M(SR,SC)=DN
2070 REM celdas vacias en la pantalla.
2080 :
2090 REM — Rutina para abrir 1 nudo:
2100 IF M(Y,X)=DN THEN RETURN
2110 IF M(Y,X)=WA THEN RETURN
2120 REM — primero hallar posicion libre:
2130 NX=0
2140 REM ** BUCLE HALLAR POSICION
2150 IF S(NN)<>DD THEN NX=NX+1:NN=NN+1
2160 IF NN>SI THEN NN=1
2170 IF NX>SI THEN PRINT "Completo!":STOP
2180 IF S(NN)<>DD THEN 2140
2190 REM — Ahora abrirlo:
2200 XY=X+Y*MW
2210 N(NN)=XY
2220 P(NN)=S
2230 S(NN)=S(S)+1
2240 H(NN)=ABS(Y-FR)+ABS(X-FC)
2250 PRINT TAB(X,Y);"+";
2260 REM lo muestra en pantalla
2270 RETURN
2280 :
2290 REM — Rutina para desandar el camino:
2300 ST=S(S)
2310 FOR Q=1 TO 10000:NEXT Q:GOSUB 1560
2320 PRINT TAB(FC,FR);C$(FO);
2330 REM ** IMPRIMIR CAMINO **
2340 S=P(S) : REM nudo padre
2350 XY=N(S) : REM coords.
2360 Y=INT(XY/MW)
2370 X=XY-Y*MW
2380 M(Y,X)=RR : REM huella de la rata!
2390 PRINT TAB(X,Y);"*";
2400 IF S>0 THEN 2330
2410 PRINT TAB(2,2);C$(RR)
2420 PRINT TAB(0,22);"Camino de ";ST;" pasos."
2430 PRINT NC;" nudos cerrados."
2440 RETURN
```

Líneas laberínticas
El laberinto está contenido en una matriz bidimensional, M(,), y las estructuras de datos para el árbol de búsqueda están retenidas en las matrices P(), S(), N() y H(). El programa combina factores de costo en curso y costo estimado hasta el objetivo, que se pueden regular alterando los valores W1 y W2 en las líneas 1080 y 1090. Tal como están listados los valores, el programa seguirá una estrategia de escalar montañas, pero quizá usted quiera experimentar. El método heurístico utilizado en este programa se basa en la *Manhattan distance*, que es de gran precisión en este ejemplo. Por consiguiente, inclinando la balanza hacia el método heurístico, haciendo que W2>W1, la búsqueda se acelerará

Complementos al BASIC

El programa está escrito para el BBC Micro. Para el C64 y el Spectrum, introducir estos cambios:

Commodore 64:
Reemplazar todos los PRINT TAB(X,Y);"MENSAJE", por P=X:Q=Y: GOSUB 3000

```
3000 :PRINT"MENSAJE"
3000 REM ** RUTINA TAB **
3010 PRINT CHR$(19);
3020 PRINT LEFT$(DW$,Q);
     TAB(P);
3030 RETURN

1040 FOR I=1 TO 25:
     DW$=DW$+CHR$(17):
     NEXT I
1490 C$(WA)=CHR$(102)
1570 PRINT CHR$(147)
```

Spectrum:
Sustituir todos los

TAB(X,Y); por AT X,Y;

```
1490 LET C$(WA)=CHR$(143)
```

Suprimir la línea 1040

	(1)	(2)	(3)	
T$()	 Perlas	 Figurillas	 Especias	Descripciones de las mercancías que ofrece el jefe.
V1()	2 p.o.	2 p.o.	1 p.o.	Precios de mercancías al zarpar el barco.
V2()	?	?	?	Precios de mercancías al regresar el barco.
CQ(,)  1 saco de sal	0.5	0.5	1	
 1 bala de tela	5	5	10	
 1 joya	3	3	6	Proporciones de trueque de las mercancías a intercambiar.
 1 cuchillo	2	2	4	

El Nuevo Mundo

Al desembarcar en el nuevo continente se producirá nuestro primer encuentro con los nativos

Al alcanzar la etapa final del juego, el viaje propiamente dicho concluye cuando el jugador consigue desembarcar en el Nuevo Mundo y comerciar con las mercancías que había adquirido al comenzar el juego. Cabe esperar que las mismas le reporten beneficios cuando regrese a puerto y venda las mercancías recientemente adquiridas. Pero el peligro no ha pasado aún: todavía hay que conocer a los habitantes de estas nuevas tierras, que pueden ser amistosos u hostiles. Por lo tanto, debe cuidarse el impedir cualquier acción que pueda provocar antagonismo entre ellos y su tripulación.

Cuando termina el bucle principal del viaje, la línea 891 llama a la subrutina 10000, que se ocupa de su llegada al Nuevo Mundo. A medida que el barco se acerca a la costa, varios grupos de nativos armados se hacen a la mar en canoas para investigar. Usted debe decidir ahora si corre el riesgo de que el barco sea atacado, negándose a disparar, o bien si abre fuego y desencadena, tal vez, una guerra.

La línea 10015 comprueba el segundo elemento de la matriz de suministros, OA(2), que indica la cantidad de armas que hay a bordo. De haber armas, debe decidir el curso de acción a seguir. La línea 10022 espera una respuesta; si digita un "sí" y emplea sus armas, morirán muchos de los nativos. Sin embargo, es probable que tal curso de acción impulse a los supervivientes a regresar al barco durante la noche y prenderle fuego, con lo cual terminaría el juego. Es obvio que no tiene sentido una agresión sin que medie provocación alguna. (Se ha insertado la línea 10044 para impedir la continuación del juego tras esta imprudente decisión.)

Si decide no disparar no habrá, por supuesto, ninguna necesidad de que los nativos devuelvan el golpe, y la línea 10026 enviará el programa a la línea 10050. En esta sección, los nativos abordan la nave de forma pacífica y dan la bienvenida a los viajeros, quienes son llevados al poblado, donde los recibe el jefe. Éste resulta ser bastante amistoso y en anteriores ocasiones ya ha realizado trueques con otros visitantes provenientes del Viejo Mundo. Después de comer y descansar, el intercambio comercial puede comenzar al día siguiente (de ello se ocupa un módulo separado). Pero, antes de que pueda iniciarse el intercambio, hemos de hacer algunos arreglos previos.

Las matrices de intercambio

Los precios de las perlas, las especias y las figurillas se determinaron cuando el barco abandonó el puerto, pero desde entonces han experimentado modificaciones.

Para facilitar el intercambio comercial, se crean varias matrices nuevas. La línea 60 Dimensiona la matriz T$(), que contiene descripciones de los tres tipos de mercancías que ofrece el jefe: perlas, figurillas y especias. La línea 61 DIMensiona una matriz, V1(), que contiene los precios de mercado de estas mercancías cuando el barco se hizo a la mar.

Las perlas y las figurillas se estaban vendiendo a dos piezas de oro cada una, y se podían comprar especias a una pieza de oro el gramo. La línea 62 DIMensiona la matriz V2(), que contiene los precios de las mercancías cuando el barco regrese a puerto. Pero como los valores de mercado de las mercancías fluctúan, un elemento aleatorio incide en la estipulación del precio final. El precio de V2(1), que representa las perlas, se establece en dos o 2 ½ piezas de oro mediante la expresión de la línea 62. En la misma línea, expresiones similares fijan el precio de las figurillas en una, dos o tres piezas de oro cada una; las especias valdrán dos o 2 ½ piezas el gramo.

Cuando se produce el intercambio comercial, las mercancías no se compran ni se venden por oro, sino que se realizan trueques según un valor de intercambio acordado entre las dos partes: un lote de mercancías a cambio de otro. El jefe ofrece al capitán del barco una cantidad de perlas, figurillas y especias por cada lote de mercancías que le ofrece el capitán. Por razones de simplicidad, daremos por sentado que los comerciantes del Nuevo Mundo poseen cantidades ilimitadas de las mercancías con que comerciarán. La cantidad de cada una se determina en función de las cantidades de cada mercancía que haya en el barco.

En las líneas 64-68 se DIMensiona una matriz bidimensional a la que se asignan valores, con las proporciones de trueque para cada producto. El primer subíndice corresponde a los cuatro artículos con los que usted comerciará: sal, tela, cuchillos y joyas. El segundo subíndice representa los tres artículos con los que comerciarán los nativos: perlas, figurillas y especias, especificándose las proporciones del trueque en tres líneas del programa.

La línea 64 establece las proporciones de trueque para sacos de sal, el primer elemento del primer subíndice. Las perlas son el primer elemento del segundo subíndice y a EQ(1,1) se le asigna un valor que da la proporción de perlas para sacos de sal. El establecimiento de EQ(1,1) en 0.5 fija la proporción de trueque en media perla por cada saco de sal. Del mismo modo, a EQ(1,2) se le asigna un valor que corresponde a la intersección del primer elemento del primer subíndice con el segundo elemento del segundo subíndice: sal y figurillas. Cuando EQ(1,2) se establece en 0.5, la proporción de trueque es una figurilla por cada dos sacos de sal. EQ(1,3)=1 establece la proporción de un gramo de especias, el tercer elemento del segundo subíndice, por cada saco de sal.

Las proporciones de trueque para el segundo elemento del primer subíndice (balas de tela) se determina en la línea 66; una bala vale 5 perlas, 5 figurillas o 10 gramos de especias. La línea 67 se ocupa del tercer elemento del primer subíndice, cuchillos, cada uno de los cuales vale tres perlas, tres figurillas o seis gramos de especias. La línea 68 se ocupa de las joyas, el cuarto elemento, y se pueden intercambiar por dos perlas, dos figurillas o cuatro gramos de especias.

En la línea 69 se DIMensiona una matriz, AO(3), para almacenar las cantidades de perlas, figurillas y especias adquiridas durante el intercambio. Ésta no debe confundirse con la matriz OA, que almacenaba la cantidad de cada mercancía adquirida al comienzo del viaje.

Módulo 11: La llegada

Dimensionamiento de las matrices del intercambio

```
60 DIM T$(3):T$(1)="PERLAS":T$(2)="FIGURILLAS":T$(3)="ESPECIAS"
61 DIM V1(3):V1(1)=2:V1(2)=2:V1(3)=1
62 DIM V2(3):V2(1)=2+(INT(RND(1)*1)/2):V2(2)=2+(INT(RND(1)*3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIM EQ(4,3)
65 EQ(1,1)=0.5:EQ(1,2)=0.5:EQ(1,3)=1
66 EQ(2,1)=5:EQ(2,2)=5:EQ(2,3)=10
67 EQ(3,1)=3:EQ(3,2)=3:EQ(3,3)=6
68 EQ(4,1)=2:EQ(4,2)=2:EQ(4,3)=4
69 DIM AO(3)
```

Adición al cuerpo principal del programa

```
890 REM LLEGADA AL NUEVO MUNDO
891 GOSUB 10000
```

Subrutina de la llegada

```
10000 REM LLEGADA AL NUEVO MUNDO
10001 PRINTCHR$(147):GOSUB 9200
10005 S$="LLEGAS AL NUEVO MUNDO*":GOSUB 9100
10006 PRINT:GOSUB 9200
10007 S$="MIENTRAS TE APROXIMAS A LA COSTA*":GOSUB 9100
10009 S$="SALEN NATIVOS EN CANOAS PARA RECIBIRTE*":GOSUB 9100
10010 PRINT:GOSUB 9200
10015 IF OA(2)=0 THEN 10050
10017 S$="SU ASPECTO ES FIERO Y ESTAN ARMADOS!!*":GOSUB 9100
10018 PRINT:GOSUB 9200
10020 S$="ABRES FUEGO? (S/N)*":GOSUB 9100
10022 INPUT I$: I$=LEFT$(I$,1)
10024 IF I$<>"N" AND I$<>"S" THEN 10022
10026 IF I$="N" THEN 10050
10028 PRINT:GOSUB 9200
10030 S$="HAN MUERTO MUCHOS NATIVOS*":GOSUB 9100
10032 S$="PERO DURANTE LA NOCHE*":GOSUB 9100
10034 S$="OTROS REGRESAN*":GOSUB 9100
10036 S$="Y LE PRENDEN FUEGO A TU BARCO!!!!*":GOSUB 9100
10038 PRINT:PRINT:GOSUB 9200
10040 S$="          JUEGO TERMINADO*":GOSUB 9100
10042 END
10044 GOTO 10042
10050 S$="TE LLEVAN A CONOCER A SU*":GOSUB 9100
10052 S$="JEFE. YA HA CONOCIDO ANTES A GENTE DE*":GOSUB 9100
10054 S$="TU RAZA Y SE MUESTRA MUY AMABLE.*":GOSUB 9100
10056 GOSUB 9200
10058 S$="LA TRIPULACION HA COMIDO Y ESTA DESCANSANDO*":GOSUB
      9100
10060 PRINT:GOSUB 9200
10062 S$="MAÑANA COMENZARA LA ACTIVIDAD COMERCIAL*":GOSUB 9100
10064 PRINT:GOSUB 9200
10066 S$=K$:GOSUB 9100
10068 GET I$:IF I$="" THEN 10068
10069 RETURN
```

Complementos al BASIC

Spectrum:

Sustituir V1() por B(), V2() por D(), EQ(,) por Q(,) y AO() por E() en todo el listado e introducir las siguientes modificaciones:

```
60      DIM T$(3,9)
10001   CLS:GO SUB 9200
10022   INPUT I$:LET I$=I$(TO 1)
10068   LET I$=INKEY$:IF I$="" THEN GO TO
        10068
```

BBC Micro:

Introducir las siguientes modificaciones:

```
10001 CLS:GOSUB 9200
10068 I$=GET$
```

Ian McKinnell

GOTO END

En este último capítulo destacaremos la necesidad de describir adecuadamente los datos en PASCAL

El diseñador del PASCAL, Niklaus Wirth, tituló uno de sus libros *Data structures+algorithms=programs* (Estructuras de datos + algoritmos = programas), que refleja la importancia que posee la descripción de los datos para formular los algoritmos que procesa esa información. Si utilizamos una matriz de chars para representar una serie de caracteres, por ejemplo, las funciones y procedimientos necesarios para procesarlos (hallar su longitud, concatenarlos, etcétera) serán sumamente diferentes de los que se requerirían si se decidiera emplear en su lugar una lista enlazada.

Tomemos a modo de ejemplo la sencilla tarea de hallar la longitud de una serie, siendo la misma desconocida. Recuerde que hemos llenado todos los elementos de matriz "libres" de la representación de matriz con caracteres ASCII NUL, chr(0), o terminado la lista dinámica con el valor de puntero NIL. La versión de matriz parece algo simple:

```
FUNCTION Longitud (S:serie):Cardinal;
  VAR
    N           :0..LongitudSerie;
    hallada     :boolean;
  BEGIN
    N:=0;
    hallada:=false;
    REPEAT
      N:=N+1;
      hallada:=S[N]=chr(0)
    UNTIL hallada OR (N=LongitudSerie);
    IF hallada
      THEN
        Longitud:=N-1
      ELSE
        Longitud:=LongitudSerie
  END; {Longitud}
```

Programa Árbol Clasificador

Este programa utiliza algunos procedimientos del anterior programa ListaCirc, siendo éstos llamados en el momento adecuado. Los datos de nombres, cantidades adeudadas y cualquier otro campo que quiera añadir se entran desde teclado y se insertan por orden alfabético ascendente en un "árbol binario". Cada nudo del árbol tiene dos punteros que lo unen a elementos "menores" o "mayores". El recorrido del árbol se realiza comparando los datos nuevos con el campo Nombre de cada nudo y tomando el campo de enlace adecuado (RamifInf o RamifSup). Cuando hallamos un nudo vacío (es decir, uno que tenga una rama de valor NIL), se inserta el elemento. La escritura de los datos en el archivo se consigue entonces de forma sencilla y natural mediante un procedimiento recursivo. Usted podría utilizar estos ejemplos como base para un potente administrador de datos: un juego de programas, quizá, diseñado a medida para su propio uso. Una advertencia: si lee un archivo ya clasificado sobre un árbol binario, cada inserción se producirá en la misma rama y el árbol se convertirá en una lista de enlace simple. Para aplicaciones más simples, probablemente lo más versátil sea la lista circular. En ella no hay punteros NIL en absoluto, ahorrándose, por tanto, una comprobación doble en cada comparación, mientras que el árbol binario tendrá (Cantidad de Nudos+1) NILs.

```
PROGRAM   ArbolClasificador (input, output, ArchivoDatos);

CONST
          NombreArchivo     = 'Arboldatos';
          LongitudSerie     = 25;

TYPE
  Cardinal          = 0..MaxInt;
  TamañoSerie       = 1..LongitudSerie;
  serie             = PACKED ARRAY [TamañoSerie] OF char;
  cosa              = RECORD
                        Nombre    :serie;
                        {otros campos..}
                        deuda     :Cardinal
                      END;  {cosa}

  TipoArchivo       = FILE OF cosa;
  arbol             = ↑rama;
                      {**referencia adelantada a:}
  rama              = RECORD
                        item            ;cosa;
                        RamifInf,
                        RamifSup        :arbol
                      END;      {rama}

VAR
  datos             : cosa;
  ArchivoDatos      : TipoArchivo;
  tronco            : arbol;

INCLUDE 'Utils.src' {**archivo fuente que contiene :SaltarBlancos,
                     LeerFicha y LeerLinea - ver programa ListaCirc**}
          {1111111111111111111111111111111111111111111}

PROCEDURE   Leer Cantidad (VAR cantidad : Cardinal);
          {**Validar un valor=Cardinal legal**}
  VAR
          OK        : boolean;
  BEGIN
    REPEAT
      REPEAT
        write ( 'Cantidad ? ': 20 );

        IF EoLn ( input ) THEN
          ReadLn ( input );


        SaltarBlancos ( input )
        {** volver a solicitarla si final linea **}
      UNTIL NOT EoLn ( input );

      LeerFicha ( input, cantidad, OK );


      IF NOT OK THEN
        WriteLn     ( '---ERROR---' : 20,
                      'por favor vuelva a entrar' )
    UNTIL OK        {**insistir en un numero valido**}

  END;      {LeerCantidad}
            {1111111111111111111111111111111111111111111}

PROCEDURE       Crecer      (VAR      hoja      : arbol;
    {**añadir al arbol**}               datos     :cosa);
  BEGIN
    new ( hoja );

    WITH hoja ↑ DO
      BEGIN
        item := datos;
```

Este algoritmo puede dar lugar a errores o a confusión. ¿Para qué necesitamos la variable booleana local hallada? Y ¿por qué debemos asignarle a Longitud el valor N−1? Éstas son molestias mínimas; pero si recurrimos a dispositivos artificiales como éste, la formulación de algoritmos más complejos puede volverse muy confusa y sujeta a errores.

La utilización de "banderas de bits" (que conocemos como tipos booleanos) es el ardid más antiguo del libro del programador, pero a menudo se introducen simplemente por el bien del ordenador y no como parte natural del algoritmo. Debe utilizarse la variable local N, no Longitud, dado que ésta es un identificador de función y no una variable. Los identificadores de función en el lado derecho de una sentencia como:

```
Longitud:=Longitud+1
```

intentarían realizar una llamada recursiva a la propia función Longitud.

Compare esto con la función Longitud necesaria para series que utilicen una lista enlazada. Con la representación dinámica, recuerde que la definición TYPE de serie es muy diferente, permitiendo la creación de una longitud de serie cualquiera. Con frecuencia, las descripciones de datos como éstas se definirán de forma recursiva.

Recursión

Muchos de los ejemplos triviales que se emplean para ilustrar la recursión se podrían expresar igualmente bien (si no mejor) como algoritmos iterativos; pero veamos la función Longitud para el tipo de serie recursiva:

```
FUNCTION Longitud (S:serie)      :Cardinal;
  BEGIN
    IF S=NIL
      THEN
          Longitud:=0
      ELSE
          Longitud:=succ(longitud(S ↑ .siguiente))
  END; {Longitud}
```

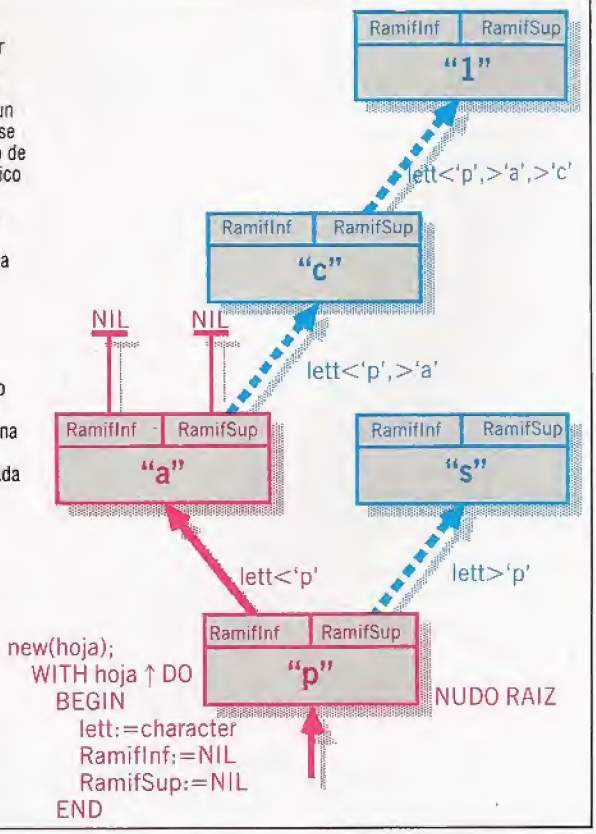
La cabeza de la lista es S y la expresión S ↑ .siguiente selecciona el campo puntero del siguiente registro de la lista. Por otra parte, se puede pensar que se trata de una lista que comienza por el siguiente registro (ButFirst). Siempre que no encontremos un NIL, llamamos a una evaluación de la longitud de ButFirst y la incrementamos con succ.

Existen muchos problemas más que sólo se pueden resolver de forma eficaz mediante el empleo natural de la recursión. El ejemplo anterior es más bien trivial y le permitirá escribir Longitud con rela-

```
    RamifInf := NIL; {**No mas datos aun,}
    RamifSup := NIL {mas alla de esta rama**}
  END

END;    {Crecer}
        {11111111111111111111111111111111111111111}

PROCEDURE       Trepar          ( brote       : arbol;
  {**hallar lugar para insertar**} datos      : cosa );
  VAR
        tallo       : arbol;
        masgrande   : boolean;
  BEGIN
    WHILE brote<>NIL DO
      BEGIN          {**hallar una rama vacia**}
        tallo := brote;
        masgrande:=datos . Nombre>
                   tallo ↑ .item . Nombre;
        IF masgrande
          THEN       {**trepar hacia arriba**}
            brote := brote ↑ .RamifSup
          ELSE       {**trepar hacia abajo**}
            brote := brote ↑ . RamifInf
      END;

    Crecer ( brote, datos );

    IF masgrande
      THEN           {**insertar arriba**}
        tallo ↑ .RamifSup := brote
      ELSE           {**insertar abajo**}
        tallo ↑ .RamifInf := brote
  END;  {Trepar}
        {11111111111111111111111111111111111111111}

PROCEDURE       Imprimir ( VAR     F       : TipoArchivo;
  {**escribir datos en un archivo**} raiz  : arbol );
  BEGIN
    IF raiz<>NIL THEN
      WITH raiz ↑ DO     {**recursion:}
        BEGIN    {**escribir por orden :}
          imprimir    ( F, RamifInf ); {**primero abajo,}
          write       ( F, item );  {la raiz misma,}
          imprimir    ( F, RamifSup ) {por ultimo abajo**}
        END

  END;  {Imprimir}
        {11111111111111111111111111111111111111111}

PROCEDURE       Reclamar ( VAR     raiz     : arbol );
  {**recuperar memoria asignada para uso ulterior**}
  BEGIN
```

```
  IF raiz<>NIL THEN {**recursion**}
    BEGIN       {**reclamar primero ramificaciones-}
      Reclamar  ( raiz ↑ .RamifInf );
      Reclamar  ( raiz ↑ .RamifSup );
      dispose   ( raiz ) {-despues la raiz**}
    END

  END;  {Reclamar}
        {11111111111111111111111111111111111111111}

BEGIN   {ArbolClasificador – Programa principal }

  asignar ( ArchivoDatos, NombreArchivo );
  pagina ( output );
  WriteLn ( '=== Clasificacion en arbol ===' : 25 );
  WriteLn;
  WriteLn ( 'Entre algunos datos ( apellido primero)' );
  write ( 'Nombre ? ' );
  LeerLinea ( datos . Nombre );
  LeerCantidad ( datos . deuda );
  Crecer ( tronco, datos ); {*plantar el tronco del arbol*}
  write ( 'Nombre ? ' );
  LeerLinea ( datos. Nombre );

  WHILE datos . Nombre [ 1 ]<>chr ( 0 ) DO
    BEGIN
      LeerCantidad ( datos . deuda );
      Trepar ( tronco, datos );
      WriteLn ( 'RETURN cuando hecho' : 40 );
      write ( 'Nombre ? ' );
      LeerLinea ( datos . Nombre)
    END;
    {**otro proceso, luego escribir el archivo :}
  WriteLn ( 'Escribir los datos en : ',Nombre Archivo );
  rewrite ( ArchivoDatos );
  Imprimir ( ArchivoDatos, tronco );
  WriteLn;
  WriteLn ( 'Deuda' : 8, 'Nombre' : LongitudSerie );
  WriteLn;
  reset     ( ArchivoDatos ); {**leer el archivo,}
            {e imprimir los datos ordenados :}
  WHILE NOT EoF ( ArchivoDatos ) DO
    BEGIN          {**leer cada registro**}
      read ( ArchivoDatos, datos );
        {**escribir los campos en la VDU : }
      WITH datos DO
        WriteLn ( deuda : 8, ' : ', Nombre )
    END;

  Reclamar ( tronco ); {**disponer de la memoria}
    {para posteriores procesos, etc.**}
END.
```

Volviendo la hoja
El programa ArbolClasificador utiliza una estructura de árbol binario para almacenar datos ordenados según el valor de un campo clave. Los elementos se insertan en la rama de abajo o de arriba, según el orden alfabético del campo del nombre. El procedimiento "Crecer" crea cada nudo nuevo y "Trepar" encuentra la ruta hasta la rama NIL correcta.
El diagrama ilustra una estructura de árbol binario simple, en el cual cada nudo retiene solamente un único carácter (*lett*) y muestra cómo se almacenaría una serie en minúsculas ("pascal"). La zona en rojo indica una etapa del crecimiento del árbol, efectuada mediante el procedimiento Crecer



Liz Dixon

tiva facilidad sin utilizar la recursión. El código es más extenso y se debe emplear un contador local (al igual que en la versión de matriz) específicamente para evitar la recursión.

```
FUNCTION Longitud (S:serie)   :Cardinal;
  VAR
    N   :Cardinal;
  BEGIN
    N:=0;
    WHILE S<>NIL DO
      BEGIN
        N:=N+1;
        S:=S ↑ .siguiente
      END;
    Longitud:=N
  END;   {Longitud}
```

Incluso en este algoritmo iterativo la claridad y la precisión son evidentes y obedecen a la simple naturaleza recursiva de la estructura de datos.

Muchos compiladores de PASCAL soportan *directivas*, que son instrucciones para el compilador, *no* declaraciones ni sentencias. En realidad la única directiva exigida por el ISO Standard es Forward (más adelante). En el caso de que dos procedimientos o funciones necesiten llamarse entre sí, se dice que son *mutuamente recursivos*. Esto se produce muy raramente, pero plantea un problema: no se puede utilizar ningún objeto del PASCAL hasta que haya sido declarado o definido.

La solución estriba en declarar sólo el encabezamiento de un subprograma, sustituyendo su bloque por la directiva del compilador FORWARD. Tras la definición completa del otro módulo, se consigna el encabezamiento de forma abreviada (omitiendo la lista de parámetros) y es entonces cuando se define el bloque. A menudo se suele disponer de otras directivas para controlar las opciones de compilación, pero no deben emplearse de forma liberal si se desea conservar la portabilidad. Esto significa que, además de que como primer carácter del comentario debe aparecer un símbolo especial (por lo general $), las opciones no portables podrán no ser aceptadas por un compilador diferente.

Más complementos

Algunas implementaciones no estandarizadas (en particular HiSoft) exigen una sintaxis ligeramente distinta para adelantar declaraciones de puntero. En este caso, usted deberá remitirse a su manual. Hay muy pocos "complementos" de esta clase, y ninguno en absoluto para los compiladores de PASCAL que se ajusten a la definición ISO del lenguaje.

Otra diferencia que quizá encuentre en las versiones UCSD, TCL/RML/Oxford y HiSoft es la falta de un procedimiento dispose. En lugar del mismo, se proporcionan los procedimientos no estandarizados mark y release.

Existe aún otra importante descripción avanzada de datos en PASCAL que aún no hemos mencionado: la "variante". Cuando hemos querido almacenar elementos de tipos diferentes, hemos utilizado un registro con campos de tipos adecuados. Pero supongamos que es necesario que la descripción de una parte, o incluso de todo el registro, sea flexible. Típicamente, puede que se desee almacenar información personal diferente sobre los llamados "sujetos de datos", que dependa, por ejemplo, de si el individuo está casado o no. En un registro variante se define primero su "parte fija", después se especifica la parte variante mediante la introducción de un selector de variante (de cualquier tipo simple) y utilizando las palabras reservadas CASE y OR:

```
TYPE
  genero=(varon,mujer);
  variante=RECORD
    {cualquier campo comun}
    CASE casado:boolean OF
      false:();
      true :(FechaCasamiento:serie;
            CASE sexo:genero OF
              varon:();
              mujer:(NombreSoltera:serie))
            END:{variante}
```

Observe que las listas de campos vacíos deben tener paréntesis. El espacio en un archivo será fijo (para la variante más larga), pero se puede guardar memoria con punteros de variantes tales como new(p,true,varon).

Los escasos puntos débiles del PASCAL en gran parte han sido eliminados mediante los esfuerzos de los especialistas en PASCAL (y por Wirth en el MODULA-2). El lenguaje es formidablemente potente y, aun así, pequeño, eficaz, para fines generales y fácil de aprender. Por supuesto, si usted se ciñe a la definición ISO habrá algunas operaciones a nivel de sistema que no podrá implementar. Una posible solución es escribir estas rutinas en ensamblador o BCPL y unirlas a un programa en PASCAL. Las ventajas de que todo su código fuente en PASCAL sea portable a cualquier micro, mini u ordenador central del mundo son indiscutibles. El PASCAL es el enfoque más próximo que tenemos a una *lingua franca* informática.

Un compatible competitivo

Chris Stevens

Alcanzando al gigante
El Tandy 1000 es una máquina compatible con el IBM-PC, con el que la empresa confía recuperar parte del mercado que le ha sido arrebatado por la intervención de la gigantesca IBM Corporation. Para conseguir la compatibilidad, el Tandy 1000 está basado en el procesador Intel 8088 y utiliza las unidades de disco estándar de 5 ¼ pulgadas. En la fotografía vemos el modelo de unidades de disco gemelas, que posee 128 K adicionales de memoria. El ordenador está visualizando el *Flight simulator* (simulador de vuelo) de Microsoft, escrito para el IBM-PC. La capacidad para ejecutar este programa es una prueba de su compatibilidad

Con la introducción del modelo 1000 la empresa Tandy espera ofrecer una alternativa relativamente económica al IBM-PC

Aunque Tandy Corporation fue uno de los primeros fabricantes que se introdujo en el mercado del microordenador, con el TRS-80, la empresa no consiguió ganar el interés masivo que obtuvo el Commodore en el mercado personal o el Apple en el mercado de gestión. Ahora Tandy parece haber modificado sus planteamientos comerciales y se dispone a un ataque bilateral sobre el mercado de gestión. Por una parte, asociándose con ACT (fabricantes del Apricot), Tandy ha puesto al día su gama de máquinas en la cadena de centros de informática de toda Europa, que están ahora vendiendo la gama Apricot. La otra punta de lanza de la ofensiva de la empresa es la que representa su propio departamento de fabricación.

A muchas personas de la industria durante cierto tiempo les ha parecido obvio que quienquiera que pudiera ofrecer una máquina verdaderamente compatible con el IBM-PC a un precio reducido, estaría llamado a ser un ganador, tanto en el mercado de gestión como en el mercado del ordenador personal de Estados Unidos, donde los consumidores tienden a adquirir máquinas para uso personal que en Europa se consideran demasiado caras para cualquier aplicación que no sea de gestión.

Se afirma que el Tandy 1000 es una máquina totalmente compatible y, con una única unidad de disco y 128 K cuesta poco más de £1 100 (unas 240 000 ptas) en el mercado británico. Sin duda alguna, Tandy confía en que, al rebajar drásticamente su precio frente al de la competencia, podrá revitalizar su departamento de productos de informática.

El ordenador que vamos a examinar aquí es la versión del Tandy 1000 de 256 K con unidades de disco gemelas. A primera vista, guarda un parecido más que casual con el IBM-PC. La máquina se compone de una caja grande que contiene el ordenador propiamente dicho, las interfaces y las unidades de disco. Encima se apoya la pantalla y hay un teclado móvil que se puede colocar del modo más conveniente.

La máquina Tandy posee el mismo aspecto sólido y fiable que el IBM-PC. Al igual que la máquina IBM, el Tandy 1000 incorpora un teclado que se enchufa al ordenador y que tiene teclas esculpidas y una superficie curvada para facilitar la digitación. Además, para contribuir a un posicionamiento óptimo, hay debajo dos patas que inclinan el teclado hasta un ángulo de 15°.

Si bien el Tandy posee las teclas necesarias para lograr la compatibilidad, éstas están dispuestas de un modo distinto al del teclado del IBM-PC. Esta estrategia tiene sus ventajas y sus inconvenientes. En el lado positivo, el teclado del IBM, aunque alabado por su diseño, ha sido categóricamente criticado en muchos sectores por una cierta inconveniencia en la colocación de algunas teclas vitales, como las teclas Shift y Alternate. El tamaño relativamente pequeño de las teclas Return y Control también ha suscitado críticas.

Es obvio que Tandy ha tomado nota de estas críticas en el momento de diseñar el teclado. La tecla "/", que causaba muchísimos problemas al utilizar la tecla Shift del lado izquierdo, ahora se ha suprimido por completo, asumiendo sus funciones las teclas de SHIFT 4 y 7 del teclado numérico. Del mismo modo, las teclas Alternate y Caps Lock se han trasladado desde sus posiciones a los lados de la barra

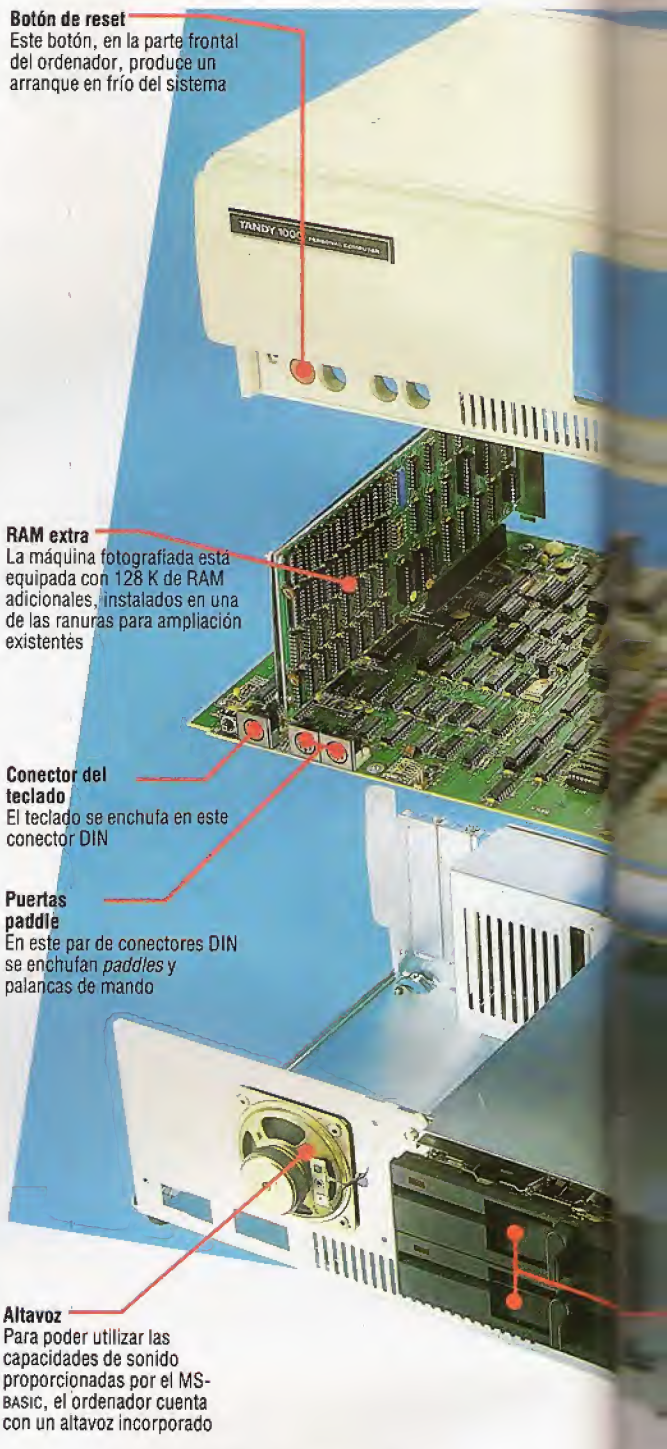
espaciadora hasta la zona del teclado numérico y abajo y a la izquierda de las teclas Control, respectivamente. Estos cambios han tenido como efecto global que la digitación normal resulte mucho más sencilla que en el teclado IBM.

La desventaja de estos cambios formales es, por supuesto, que una vez que uno se ha tomado la molestia de familiarizarse con el idiosincrásico trazado del IBM-PC, tendrá que volver a aprenderse el teclado. No obstante, a los recién iniciados les resultará mucho más fácil habituarse al mismo. Las teclas son de recorrido total y su calidad "táctil" es una de las más estimables del mercado.

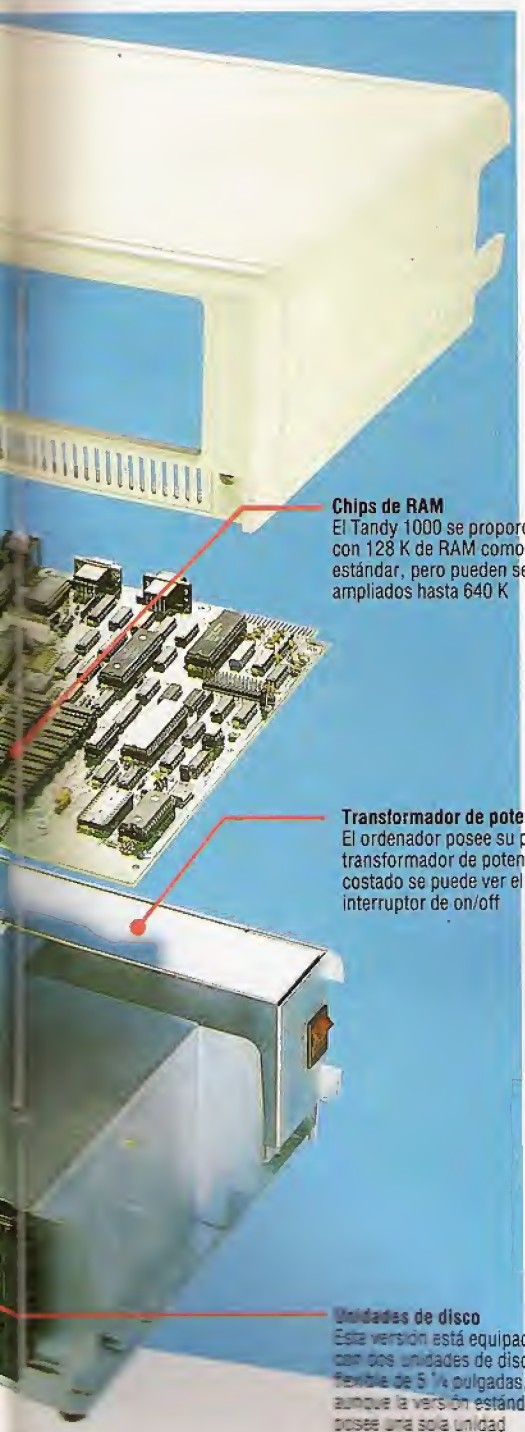
Asimismo, Tandy ha introducido otros cambios en el teclado. Las teclas Insert y Delete se han incorporado con las funciones + y − y ahora se hallan encima del teclado, no debajo. Entre las adicionales se incluyen teclas individuales del cursor para edición, así como teclas Hold, Print y Break, todas las cuales se han situado entre las de máquina de escribir y las de teclado de calculadora. La adición de estas teclas ha supuesto la reubicación de las teclas Function, desde el extremo izquierdo del teclado en el IBM-PC, hasta justo encima de las de máquina de escribir. Aunque probablemente haya sido inevitable, ahora las teclas están situadas de forma menos conveniente que lo que lo estaban antes, pero, a modo de compensación, Tandy ha proporcionado dos teclas Function adicionales.

Retornando al ordenador propiamente dicho, Tandy ha optado por la forma de caja grande propiciada por IBM. En la parte frontal, en el lado izquierdo, hay un par de unidades de disco flexible de 5 ¼ pulgadas, si bien el modelo estándar posee una sola unidad pero con facilidad para albergar otra. Aunque en líneas generales son algo más silenciosas que sus equivalentes IBM, las unidades mostraron pequeñas diferencias en cuanto a velocidad de acceso. Una molestia mínima es que los discos no se cargan a resorte. Esto significa que no saltan fuera de las unidades cuando éstas se destraban y que hay que sacarlos con los dedos.

Debajo de la proyección frontal, junto a una rejilla de ventilación, hay un par de enchufes DIN de 270° y seis patillas para palancas de mando u otros dispositivos de control externo. A la izquierda de éstos hay un botón grande de color naranja, la reinicialización del sistema, que proporciona el arranque en frío. A pesar del hecho de estar tan a la vista, existen muy pocas probabilidades de pulsarlo



Chris Stevens



inadvertidamente, puesto que se halla en la carcasa. De hecho, quizá sea la mejor posición posible, porque numerosas aplicaciones no permiten que usted salga al sistema operativo sin pulsar antes este botón. Por consiguiente, resulta útil un *reset* situado convenientemente, en especial en una máquina tan grande.

En la parte trasera del ordenador están las interfaces para periféricos. En el extremo izquierdo se halla la entrada de la fuente de alimentación eléctrica, justo encima de la interface en paralelo Centronics, proporcionada como conector marginal. A la derecha hay un conector D de siete patillas para un lápiz óptico y una puerta para la conexión de un monitor RGB. Más hacia la derecha, debajo del ventilador de refrigeración, hay un par de conectores microjack, uno de los cuales proporciona un conector de video compuesto para pantallas monocromáticas o a color de video compuesto; el otro es un conector de audio que puede amplificar el sonido del ordenador a través de un sistema de alta fidelidad convencional. En el extremo derecho hay tres conectores de ampliación en los que se pueden instalar interfaces para periféricos adicionales o placas de memoria.

Méritos comparativos

Tandy ha hecho hincapié en el hecho de que su máquina proporciona muchas facilidades de interface de las que no dispone el IBM-PC estándar. Aunque esto es verdad, la empresa parece haber proporcionado apenas el mínimo necesario para una máquina de gestión utilizable. Por ejemplo, la máquina de precio mínimo está dotada de 128 K de RAM, lo que no es suficiente para ejecutar algunos de los paquetes de software integrado más nuevos. Y, si bien el Tandy 1000 es una máquina completamente autosuficiente, una puerta RS232 (que es opcional) es esencial para cualquier clase de comunicaciones que pueda requerir el usuario de gestión. Por otra parte, ciertamente ofrece, considerando su precio, más prestaciones que el IBM-PC, el cual, con las mismas facilidades, costaría mucho más. Siempre y cuando, por supuesto, el Tandy sea auténticamente compatible con el software IBM.

El problema de intentar la compatibilidad con el IBM-PC no estriba ni en la CPU 8088 y ni siquiera en el sistema operativo MS-DOS, dado que ambos se pueden adquirir contactando con los fabricantes adecuados. La dificultad estriba en el BIOS (Basic Input/Output System: sistema básico de entrada/salida), cuyo *copyright* pertenece a IBM. Muchos programas emplean saltos directos a las rutinas BIOS y, a menos que una máquina compatible tenga las rutinas relevantes en las mismas direcciones, el programa no funcionará correctamente.

En este sentido, el BIOS del Tandy 1000, escrito por Phoenix Compatibility Corporation, es excepcional. En la máquina no sólo se ejecuta el *Lotus 1-2-3*, cuyas dificultades son notorias (aunque aún con 256 K de espacio de memoria estaba bastante justo), sino que el ordenador también ejecuta el *Wordstar 2000* y el *dBase II*.

A pesar de ello, el Tandy 1000, al igual que el IBM-PC, es sumamente lento para cálculos matemáticos en BASIC. Completar un contador de 0 a 1000, utilizado como sencillo programa comparativo, llevó seis segundos completos, más que muchos micros de ocho bits.

El Tandy 1000 es, indudablemente, una alternativa viable para el usuario de gestión que desee sacar partido de la gran cantidad de software de calidad que se ha escrito para el IBM-PC. Sin embargo, debido a las limitaciones de hardware que impone la compatibilidad con IBM y a la necesidad de mantener el precio lo más bajo posible, Tandy ha presentado una máquina que, desde el punto de vista de la informática de gestión, es esencialmente anticuada. Así y todo, siempre y cuando IBM no reduzca el precio de su máquina a modo de represalia (lo que parece improbable), la Tandy Corporation habrá producido una máquina que obtendrá un enorme éxito.

TANDY 1000

DIMENSIONES
420×335×150 mm

CPU
8088, operando a 4,77 MHz

MEMORIA
128 K de RAM como estándar; ampliables a 640 K

PANTALLA
80×25 caracteres en modalidad de texto, 640×200 pixels en alta resolución. La máquina puede visualizar hasta ocho colores entre una gama de 16

INTERFACES
Conectores para monitor compuesto y RGB, puerta para impresora, enchufe de audio y tres interfaces para ampliación

LENGUAJES DISPONIBLES
BASIC, más una selección disponible bajo MS-DOS incluyendo COBOL, FORTRAN, etc.

TECLADO
90 teclas en total, incluyendo un teclado numérico y 12 teclas de función

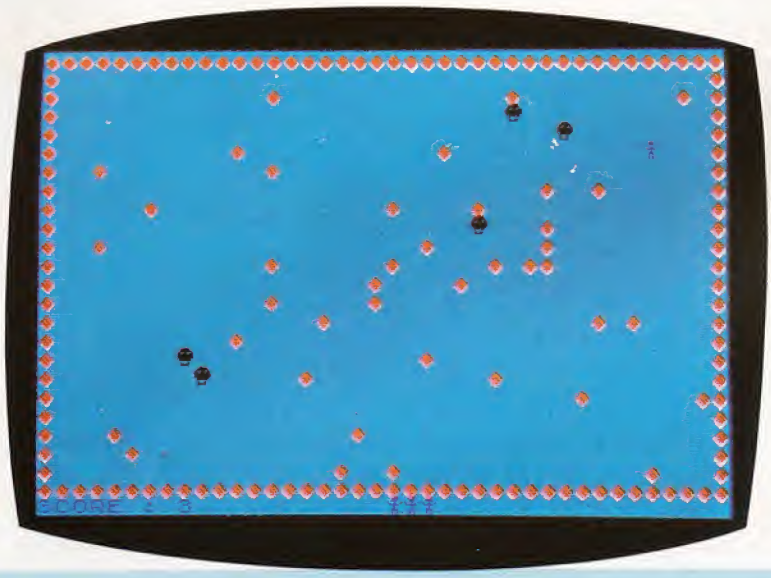
DOCUMENTACION
La documentación incluye explicaciones del MS-DOS, MS-BASIC y el paquete integrado Deskmate que se entrega junto con la máquina

VENTAJAS
Totalmente compatible con el IBM-PC, proporcionando una cantidad de interfaces de las que no dispone el modelo IBM básico

DESVENTAJAS
Con el diseño de la máquina, Tandy no ha avanzado más allá del diseño del IBM-PC original. La utilización de un procesador 8088 en lugar del 8086, más avanzado, significa que la operación del Tandy 1000 es más lenta de lo que cabía esperar

Robots en el MO5

Usted se encuentra solo, abandonado sobre un planeta defendido por robots asesinos. El suelo está sembrado de minas... Ya conoce el problema. Pero tal vez no en el MO5 de Thomson



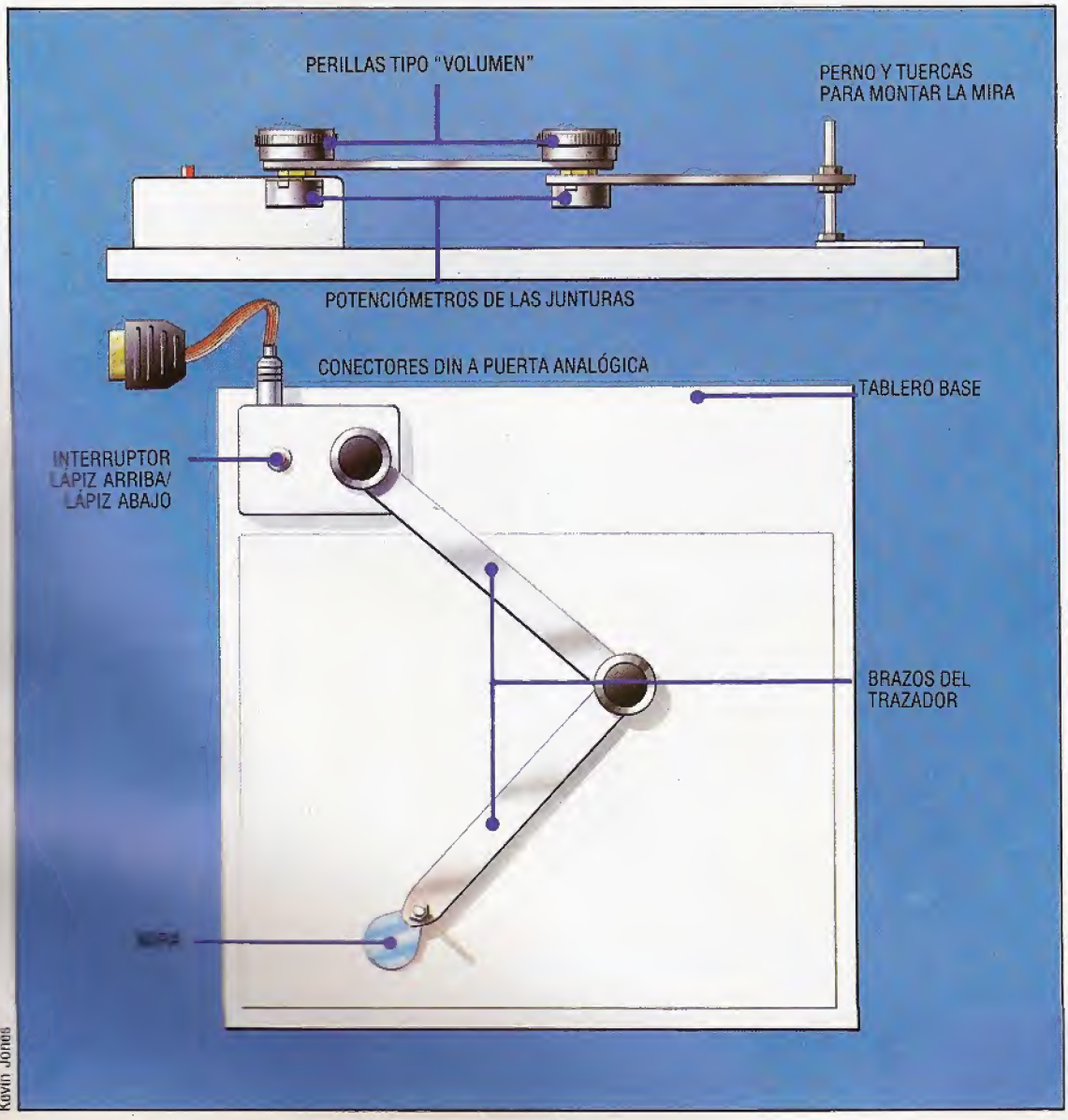
Las minas están representadas en la pantalla por rombos rojos. Al comenzar el juego, hay cinco robots preparados sobre el terreno. Sin perder un solo segundo, se precipitan sobre usted siguiendo siempre el camino más corto. Afortunadamente, son ciegos y no pueden ver las minas situadas entre usted y ellos, lo cual le permitirá, siempre que se desplace de forma adecuada, eliminarlos. Para ello utilice la palanca de mando o las teclas A, Z, E, Q, D, W, X, C, según la dirección elegida por usted. Una vez eliminados todos los robots, el juego prosigue con un robot suplementario. Si salta sobre una mina o un robot le mata, aún no está todo perdido. Usted dispone de cinco vidas. Si desea cambiar el número de minas, modifique el valor de la variable NM en la línea 80.

```
10 REM **************
20 REM *  ROBOTS  *
30 REM **************
40 DEFINT A-Z
50 CLEAR ,,3
60 NH=5
70 N1=5
80 NM=40
90 NR=N1
100 DIM R(30,1)
110 GOSUB 1580
120 GOSUB 1470
130 GOSUB 910
140 ON JS GOSUB 710,810
150 C=POINT(HX*8+4,HY*8+4)
160 IF C<>-13 AND C<>4 THEN 470
170 COLOR 4
180 LOCATE X,Y
190 PRINT N$;
200 LOCATE HX,HY
210 PRINT H$;
220 X=HX
230 Y=HY
240 T=0
250 FOR I=1 TO NR
260 IF R(I,0)=0 THEN 400
270 T=1
280 RX=R(I,0)+SGN(HX-R(I,0))
290 RY=R(I,1)+SGN(HY-R(I,1))
300 C=POINT(RX*8+4,RY*8+4)
310 IF C=1 OR C=0 THEN S=S+1:LOCATE R(I,
    0),R(I,1):PRINT N$;:R(I,0)=0:GOTO 400
320 IF C=4 THEN 470
330 COLOR 0
340 LOCATE R(I,0),R(I,1)
350 PRINT N$;
360 LOCATE RX,RY
370 PRINT R$;
380 R(I,0)=RX
390 R(I,1)=RY
400 NEXT I
410 IF T=0 THEN 430
420 GOTO 140
430 S=S+10
440 IF INKEY$<>"" THEN 440
450 IF NR<30 THEN NR=NR+1
460 GOTO 130
470 NH=NH-1
480 COLOR 7
490 LOCATE X,Y
500 PRINT N$;
510 LOCATE HX,HY
520 PRINT H$;
530 PLAY "L96REL72REL24REL96REL72FAL24MI
    L72MIL24REL72REL24DO#L96RE"
540 IF INKEY$<>"" THEN 540
550 IF NH>0 THEN NR=N1:GOTO 130
560 CLS
570 SCREEN 1,6,6
580 ATTRB 1,1
590 LOCATE 9,10
600 PRINT "PUNTOS :";S;
610 LOCATE 9,20
620 PRINT "OTRA ?";
630 COLOR 4
640 ATTRB 0,0
650 IF INKEY$<>"" THEN 650
660 D$=INKEY$
670 IF D$="" THEN 660
680 IF D$<>"N" THEN RUN
690 CLS
700 END
710 D$=INKEY$
720 IF D$="A" THEN HX=HX-1:HY=HY-1
730 IF D$="Z" THEN HY=HY-1
740 IF D$="E" THEN HY=HY-1:HX=HX+1
750 IF D$="Q" THEN HX=HX-1
760 IF D$="D" THEN HX=HX+1
770 IF D$="W" THEN HX=HX-1:HY=HY+1
780 IF D$="X" THEN HY=HY+1
790 IF D$="C" THEN HY=HY+1:HX=HX+1
800 RETURN
810 J=STICK(0)
820 IF J=1 THEN HY=HY-1
830 IF J=2 THEN HY=HY-1:HX=HX+1
840 IF J=3 THEN HX=HX+1
850 IF J=4 THEN HX=HX+1:HY=HY+1
860 IF J=5 THEN HY=HY+1
870 IF J=6 THEN HY=HY+1:HX=HX-1
880 IF J=7 THEN HX=HX-1
890 IF J=8 THEN HX=HX-1:HY=HY-1
900 RETURN
910 CLS
920 COLOR 4
930 LOCATE 0,24
940 PRINT "PUNTOS :";S;
950 IF NH=1 THEN 1000
960 FOR HX=1 TO NH-1
970 LOCATE 19+HX,24
980 PRINT H$;
990 NEXT HX
1000 COLOR 1
1010 FOR HX=0 TO 39
1020 LOCATE HX,0
1030 PRINT M$;
1040 LOCATE HX,23
1050 PRINT M$;
1060 NEXT HX
1070 FOR HY=1 TO 22
1080 LOCATE 0,HY
1090 PRINT M$;
1100 LOCATE 39,HY
1110 PRINT M$;
1120 NEXT HY
1130 FOR I=1 TO NM
1140 HX=INT (RND*38)+1
1150 HY=INT(RND*22)+1
1160 IF SCREEN(HX,HY)<>32 THEN 1140
1170 LOCATE HX,HY
1180 PRINT M$;
1190 NEXT I
1200 COLOR 0
1210 FOR I=1 TO NR
1220 R(I,0)=INT(RND*38)+1
1230 R(I,1)=INT(RND*22)+1
1240 IF SCREEN(R(I,0),R(I,1))<>32 THEN
     1220
1250 LOCATE R(I,0),R(I,1)
1260 PRINT R$;
1270 NEXT I
1280 HX=INT (RND*38)+1
1290 HY=INT (RND*22)+1
1300 IF SCREEN(HX,HY)<>32 THEN 1280
1310 X=HX
1320 Y=HY
1330 FOR I=1 TO 5
1340 LOCATE HX,HY
1350 COLOR 5
1360 PRINT CHR$(127);
1370 BEEP
1380 FOR J=1 TO 50
1390 NEXT J
1400 LOCATE HX,HY
1410 COLOR 4
1420 PRINT H$;
1430 FOR J=1 TO 50
1440 NEXT J
1450 NEXT I
1460 RETURN
1470 CLS
1480 SCREEN 4,12,0
1490 ATTRB 1,1
1500 LOCATE 10,10,0
1510 PRINT "PAL. MANDO ?";
1520 ATTRB 0,0
1530 D$=INKEY$
1540 C=RND
1550 IF D$="" THEN 1530
1560 IF D$="0" THEN JS=2 ELSE JS=1
1570 RETURN
1580 DEFGR$(0)=28,28,73,62,8,28,20,20
1590 DEFGR$(1)=60,126,219,255,255,126,36,60
1600 DEFGR$(2)=0,0,24,60,126,126,60,24
1610 H$=GR$(0)
1620 R$=GR$(1)
1630 M$=GR$(2)
1640 N$=CHR$(32)
1650 RETURN
```

Nuevo proyecto

Iniciamos la construcción de un trazador digital para utilizar con el BBC Micro. En primer lugar, esbozamos las etapas del proyecto y proporcionamos la lista de componentes

Un trazador digital es, simplemente, un dispositivo mediante el cual se pueden trazar formas y figuras reales sobre un tablero y visualizarlas en una pantalla de ordenador. Una vez en forma digital, una imagen se puede ya sea editar o bien guardar en disco o cinta. El componente básico del dispositivo es un brazo metálico, dividido en dos partes y con unos goznes para conformar una juntura de hombro y codo. Sobre estos goznes hay montados dos potenciómetros que se pueden usar para proporcionar datos sobre los ángulos de cada juntura. Mediante el empleo del convertidor A/D que lleva incorporado el BBC Micro, la salida analógica de cada potenciómetro se puede convertir a forma digital y, tras el calibrado, los datos entrantes se pueden manipular luego matemáticamente para dar la posición del extremo del segundo brazo en relación a la juntura del hombro. Las figuras de las ilustraciones muestran cómo encajan entre sí los elementos básicos de construcción. La unidad se asienta sobre un tablero cuadrado de 46 cm en el cual se puede colocar la figura a trazar. Los brazos de metal se montan sobre una pequeña caja plástica, en la cual va montado, asimismo, un conector DIN



Kevin Jones

Elementos de trazado
El trazador digital de *Bricolaje* utiliza dos potenciómetros para proporcionar información que se pueda convertir mediante software, merced a la puerta analógica del BBC Micro, para dar la posición de la mira sobre el tablero

de 5 patillas que acepta un cable conector a la puerta analógica del ordenador, así como un interruptor de presión que le proporciona al trazador una opción incorporada de lápiz arriba/lápiz abajo. En la punta del trazador se fija un pequeño trozo de plástico transparente, en cuya superficie hay grabadas señales en cruz para conformar una mira. El sistema de montaje está diseñado de modo tal que la mira se pueda subir o bajar para facilitar el trazado exacto sobre materiales de espesor variable.

Lista de componentes

Cant.	Artículo
2	Potenciómetro lineal de 100 K-ohmios
2	Perilla de 40 mm
1	Interruptor de presión
1	Caja de 114×76×38 mm
1	Enchufe DIN de 5 patillas
1	Conector DIN de 5 patillas
1 m	Cable plano de 10 vías
1	Caja pernos M5
1	Caja pernos M3
1	Caja tuercas M3
1	Caja tornillos sin tuerca
1	Enchufe D de 15 vías
1	Cubierta D de 15 vías

Varios

1	Fleje de aluminio de 600×25×2,5 mm*
1	Tablero recubierto de melamina de 46×46 mm**
1	Trozo pequeño de plástico transparente***

Observaciones

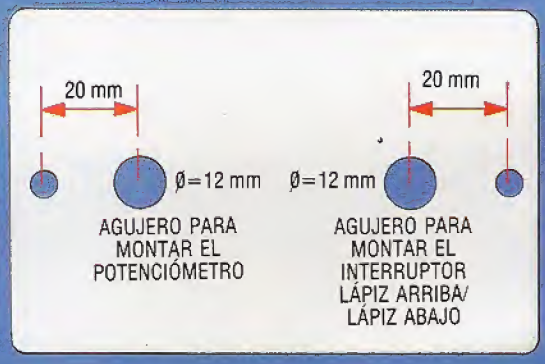
* A adquirir en la mayoría de ferreterías. Se podrían usar también tiras de madera

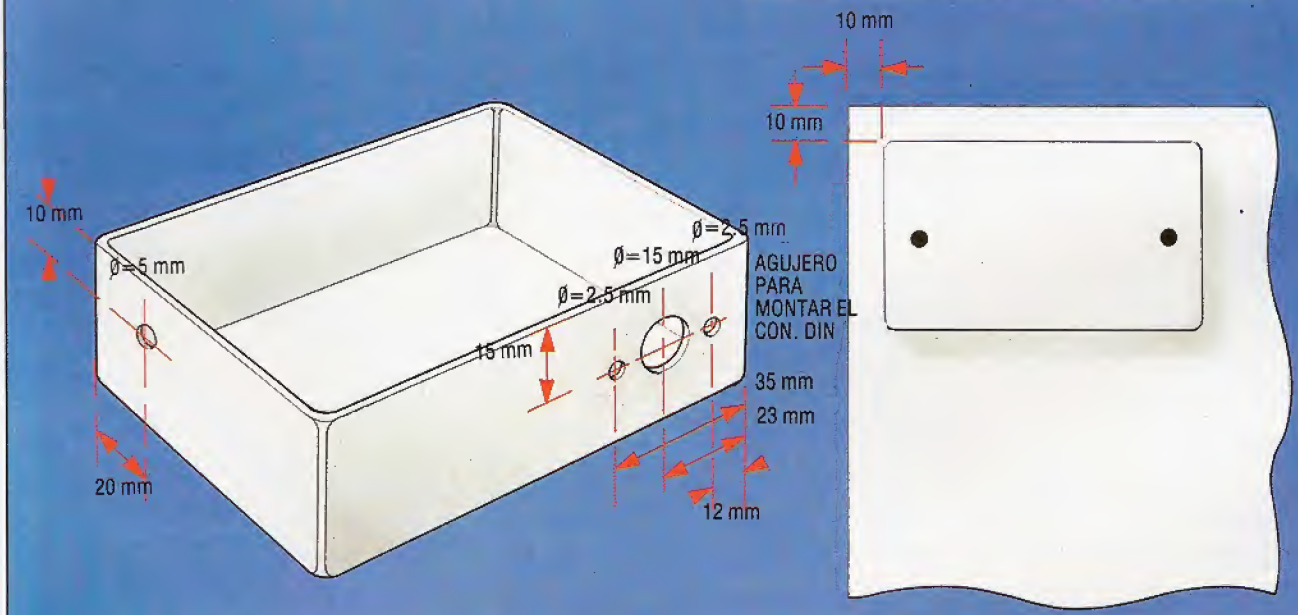
** También llamado "Contiplas", se puede asimismo adquirir en ferreterías en diversas anchuras y longitudes

*** Se puede utilizar la sección transparente de un estuche de cassette

Paso 1: Taladrado de la caja

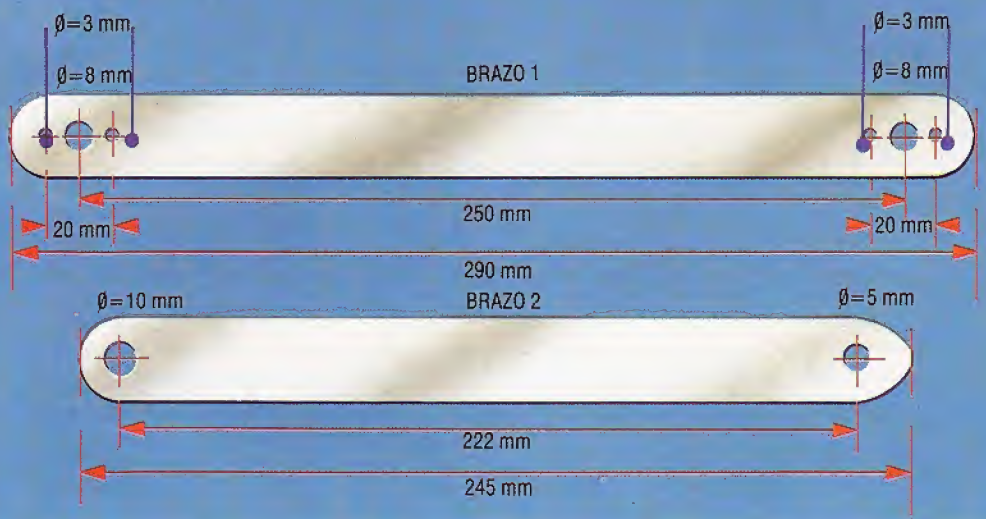
La primera etapa en la construcción del trazador es, si fuera necesario, cortar el tablero de base al tamaño adecuado. Luego se habrán de perforar en la pequeña caja plástica una serie de agujeros en los que se instalarán, como se aprecia en la ilustración, el brazo del trazador, el interruptor y el conector DIN. Tras haber perforado todos estos agujeros, se puede montar la caja en la esquina superior izquierda del tablero base, dejando un margen libre de 10 mm desde los bordes





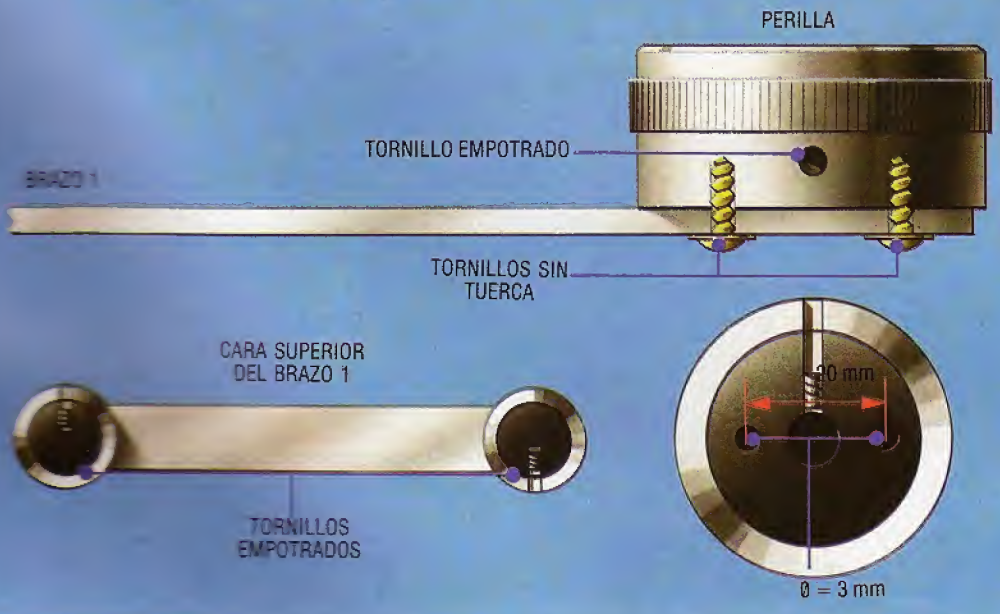
Paso 2: Cortado del brazo

A continuación, corte el fleje metálico formando las dos longitudes de la ilustración y taladre los agujeros requeridos. Observe, sin embargo, que si bien no son imprescindibles las longitudes exactas, sí deben ser precisas las distancias entre los agujeros taladrados en cada extremo de los brazos. Ello se debe a que estas distancias las utilizará el software para llevar a cabo los diversos cálculos geométricos necesarios para determinar la posición de la mira del trazador cuando se use la unidad. Por consiguiente, es importante medir cuidadosamente estas distancias antes de comenzar a perforar



Paso 3: Colocación de las perillas

La última tarea de este capítulo consiste en montar la perilla de "volumen" en cada uno de los extremos del brazo 1. Estas perillas se utilizarán para fijar los potenciómetros que implementaremos en el próximo capítulo. Observe que el husillo del potenciómetro se sujetará mediante un pequeño tornillo empotrado en el costado de la perilla. Se ha de tener cuidado al montar las perillas, de modo que los tornillos empotrados queden orientados en dirección opuesta. Las perillas se fijan en el brazo perforando dos agujeros de 3 mm de diámetro en la base de cada perilla, atravesándolas casi totalmente. Luego se deben instalar las perillas en el brazo 1 con la orientación correcta, utilizando tornillos sin tuerca, como se indica



Kevin Jones

Entramados

Estudiaremos en dos capítulos algunas rutinas de coma flotante del intérprete de BASIC en el Commodore 64

Las rutinas de coma flotante del intérprete de BASIC no están muy bien documentadas (no hay, por ejemplo, una tabla de saltos como la hay para el núcleo), por lo que la búsqueda de las llamadas puede ser una tarea lenta a base de intentos. Para dar a nuestra investigación una meta vamos a tratar de crear una rutina de gráficos en código máquina que nos permita obtener un dibujo en tres dimensiones por medio de líneas y sobre pantalla en alta resolución.

Las ideas matemáticas que aquí exponemos pueden emplearse como base de otras rutinas aritméticas más rápidas, como sería el producto de matrices. Se ha de recordar, desde luego, que los cálculos aritméticos en tiempo real para crear imágenes visuales sucesivas no constituyen necesariamente el mejor enfoque. A menudo es preferible calcular previamente los datos a partir de los cuales se construirán las figuras, antes de comenzar la secuencia del movimiento. Aun así, la técnica en tiempo real encaja con nuestros propósitos mejor que otra.

Las variables del BASIC se almacenan en la memoria encima del programa en BASIC. Estas variables se contienen en una tabla de variables, a cuyo inicio apunta el contenido de las posiciones 45 y 46 (decimal). Como todo puntero del Commodore 64, ambos están almacenados en la forma *lo-hi*. Por ello la dirección de inicio de la tabla de variables queda apuntada por la fórmula:

PEEK(45)+256*PEEK(46)

En el siguiente cuadro damos los distintos punteros asociados con las variables y su contenido normal:

Puntero	Función	Contenido normal
43/44	Inicio del BASIC	2049
45/46	Inicio de las variables	Depende de la longitud del prog.
47/48	Inicio de las tablas	Depende del núm. de variables
49/50	Fin de las tablas+1	Depende del núm./tamaño tablas
51/52	Vars. serie act. parte inf.	Depende del núm./long. variables
55/56	Parte sup. de la mem.	40960

Obsérvese que las variables dinámicas en serie se disponen a partir de la parte superior de la memoria según se van definiendo. Pero las tablas se almacenan por encima de las demás variables en la tabla de variables, y cuando en el curso de la ejecución del programa se encuentra una nueva variable, el OS desplaza toda el área de almacenamiento de tablas hacia arriba en el número de bytes necesario para almacenar la nueva variable.

El almacenamiento de variables en serie es por fuerza más complejo que el del resto de las variables. Las variables enteras (no ligadas en una tabla) y las variables de coma flotante sólo necesitan 7 bytes cada una, pero una variable en serie puede necesitar hasta 255 bytes. Para eludir esta complicación, el OS sólo almacena la longitud de la variable y un puntero que apunte a su dirección de inicio dentro de la tabla de variables. Si se define una variable literal en un programa BASIC (p. ej., A$="AMOR"), ésta apuntará al primer byte de la variable alfanumérica dentro del área para el programa BASIC. Tal variable se denomina *estática*. En el momento en que el programa altera su valor pasa a llamarse *dinámica*. Los valores de las variables alfanuméricas dinámicas se construyen a partir de la parte superior de la memoria cambiándose el valor del puntero en la tabla de variables. De esta manera toda entrada se mantiene en 7 bytes constantes.

En una tabla de variables podemos encontrar fácilmente la dirección de una variable desde el BASIC mediante un ligero trabajo de rastreo. El quid de la cuestión radica en que cada vez que el intérprete BASIC llame a una variable, su dirección vendrá apuntada por un puntero de la página cero situado en las direcciones decimales 71 y 72, que puede ser examinado para descubrir la dirección buscada. Pero hemos de apresurarnos en recabar esta información dado que la posición 71 también la emplea el sistema operativo cuando debe evaluar expresiones numéricas.

El programa que sigue ilustra el formato de variables ordinarias según son almacenadas en la memoria. La primera rutina recupera una variable alfanumérica de la memoria. Aquí empleamos la técnica que acabamos de describir para inspeccionar el contenido de las posiciones 71 y 72, almacenando inmediatamente estos valores en dos posiciones de reserva dentro del buffer para cassette. Serán empleadas posteriormente para calcular la dirección de la variable en serie y recuperarla.

```
1000 REM**BUSQUEDA DE VAR SERIE EN MEMORIA**
1010 X$="ABCDEF"
1020 REM**HACE X$ VARIABLE ACTUAL**
1030 X$=X$+" "
1040 REM**GUARDA PUNTERO DE TABLA VAR**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**DIRECCION EN TABLA VAR**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**MIRA ENTRADA EN TABLA VAR**
1090 LS=PEEK(ADR):REM LONGITUD DE VAR SERIE
1100 SA=PEEK(ADR+1)+256*PEEK(ADR+2)
1110 REM SA ES LA DIR INICIO DE VAR SERIE
1120 REM**AHORA LEE LA VAR SERIE**
1130 FORI=SATOSA+LS
1140 VAR$=VAR$+CHR$(PEEK(I))
1150 NEXT
1160 PRINTVAR$
```

Puede que se le ocurra pensar que si empleamos variables enteras (indicadas con %, como X%) ahorraremos memoria y aceleraremos las operaciones aritméticas. Pero no es así en el Commodore 64. Cuando esta máquina tiene que realizar operaciones aritméticas con números enteros, los convierte a coma flotante y llama a las rutinas de coma flotante. Por eso, aunque las variables enteras se pue-

den almacenar en sólo dos bytes, se les reserva siete bytes de memoria si no están almacenadas en una tabla. Estos bytes extra serán ignorados en el curso del proceso.

La siguiente rutina localiza en la memoria una variable entera.

```
1000 REM**LOCALIZACION EN MEM. DE VAR ENTERA**
1010 X%=3456
1020 REM**HACE X% VARIABLE ACTUAL**
1030 X%=X%
1040 REM**GUARDA PUNTERO DE TABLA VAR**
1050 POKE828,PEEK(71):POKE829,PEEK(72)
1060 REM**DIRECCION EN TABLA VAR**
1070 ADR=PEEK(828)+256*PEEK(829)
1080 REM**MIRA ENTRADA EN TABLA VAR**
1090 LO=PEEK(ADR+1):HI=PEEK(ADR)
1100 REM**CALCULA RESULTADO**
1110 SIGNBIT=(HIAND128)/128
1120 VAR=LO+256*(HIAND127)-32768*SIGNBIT
1130 PRINT VAR
```

Esta misma técnica es válida para una variable de coma flotante. Hay, sin embargo, un método más económico. Se basa en que al servirnos de DEF FN para definir una función de la variable X, se emplea X (sin que cambie su valor) siempre que se llama a FN.

Este otro programa emplea DEF FN para calcular la dirección de la variable BASIC actualmente contenida en las posiciones 71 y 72. Como X sirve de variable de la función, estamos seguros de generar la dirección que pertenece al primer byte de X contenido en la tabla de variables del BASIC. Después se llama a FN para asignar esta dirección a la variable ADD. Obsérvese que el poner a cero (u otro valor) por medio de la instrucción FN no cambia el valor de X contenido en la tabla de variables, ni la dirección calculada por la función.

```
1000 REM**PARA HALLAR UN VAR DE CF EN MEMORIA**
1010 DEF FNADR(X)=PEEK(71)+256*PEEK(72)
1020 ADD=FNADR(0):REM DA SIEMPRE LA DIRECCION DE X
1030 X=-3.14159
1040 REM**CONVERSION BIN A DECIMAL**
1050 POWERTWO=2^(PEEK(ADD)-129)
1060 SIGN=(-1)^((PEEK(ADD+1)AND128)/128)
1070 REM*LA FRACCION ES DE 31 BITS*
1080 D1=PEEK(ADD+1)AND127:REM 7 BITS
1090 D2=PEEK(ADD+2):REM 8 BITS
1100 D3=PEEK(ADD+3):D4=PEEK(ADD+4)
1110 REM**GLUP**
1120 FRACT=2^(-7)*D1+2^(-15)*D2+2^(-23)*D3+2^
     (-31)*D4
1130 MANT=1+FRACT
1140 VAR=SIGN*POWERTWO*MANT
1150 PRINTVAR
```

Tablas en la memoria

La ejecución de la instrucción DIM hace que se reserve espacio de memoria para una tabla. Se compone de un byte de cabecera más todos los demás bytes necesarios para almacenar el elemento. El formato de los elementos almacenados en una tabla difiere según el tipo de variable que acabamos de ver. El cuadro adjunto resume estas variantes. Hay que comprender bien estas diferencias si se desea acceder a los elementos de una tabla desde código máquina.

Pero antes veamos algunos aspectos de las operaciones aritméticas con coma flotante. Cuando el intérprete del BASIC está realizando algún cálculo en coma flotante almacena los resultados intermedios en dos *acumuladores de coma flotante*. Se los denomina, respectivamente, FAC y ARG, y el formato que se emplea es el mismo que para el almacenamiento de variables en la memoria. FAC se encuentra en las direcciones $61 hasta la $65 (97 y 101 en decimal) y ARG ocupa de la $69 a la $6D (105 y 109). Para mayor sencillez continuaremos usando las rutinas del intérprete que sólo intercambian números entre FAC y la memoria.

Las rutinas del intérprete con las que trabajaremos en este apartado son:

- MOVFM (llamada en dirección $BBA2):
Esta rutina sirve para cargar el contenido de FAC desde una variable en coma flotante a la memoria. Se representa simbólicamente así: F←M. Para llamarla se cargará el acumulador con el byte *lo* de la dirección de inicio de la variable en memoria, y el registro Y con el byte *hi*.

- MOVMF (llamada en dirección $BBD4):
Esta rutina coloca el contenido de FAC en siete bytes de la memoria: M←F. Para llamarla se cargará el registro X con el byte *lo* y el registro Y con el byte *hi* del byte de inicio del destino de la variable en memoria.

- FMULT (llamada en dirección $BA28):
Es la rutina de la multiplicación, por la cual se multiplica el contenido de FAC por el valor de una segunda variable en memoria, y se almacena el resultado en FAC. La primera variable se coloca en FAC por medio de MOVFM y se apunta a la segunda variable cargando el acumulador con el byte *lo* y el registro Y con el byte *hi* del byte de inicio, antes de llamar a esta rutina. Finalmente, si es necesario, podemos devolver el resultado a la memoria empleando MOVMF.

- FADD (llamada en dirección $B867):
Esta rutina de suma realiza esto: FAC=MEM-FAC. Para llamarla, hay que cargar el acumulador con el byte *lo* de MEM y el registro Y con el byte *hi* de MEM.

- FSUB (llamada en dirección $B850):
Esta rutina de resta realiza esto: FAC=MEM-FAC. Para llamarla cargamos el acumulador con el byte *lo* de MEM y colocamos el byte *hi* de MEM en el registro Y.

Éstas son las rutinas de intérprete que incorporaremos en la primera fase de nuestro programa de gráficos. Examinemos otros componentes de dicho programa.

La idea subyacente de este proyecto de gráficos es que la figura reticular de líneas puede especifi-

Variables en memoria

Variables enteras

Byte *hi* entero	Byte *lo* entero
Byte 0	Byte 1

Variables de coma flotante

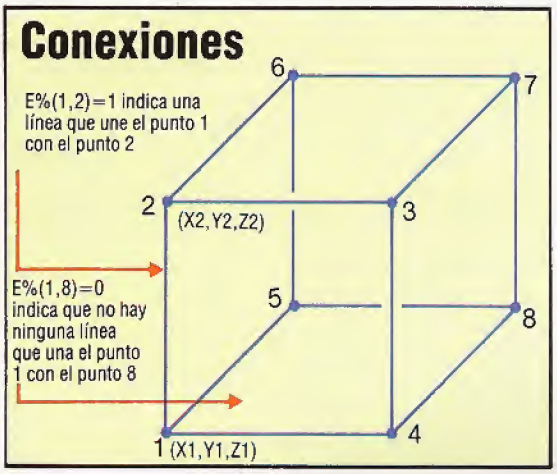
Exponente binario más 128	Bit del signo And. Mantisa 1 (7 bits)	Mantisa 2	Mantisa 3	Mantisa 4
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4

Variables en serie

Número de caracteres	Dirección inicio variable byte *lo*	Dirección inicio variable byte *hi*
Byte 0	Byte 1	Byte 2

Tipos de variables
Existen tres tipos de variables de tablas en el Commodore 64, cada una con un formato distinto en memoria. Las variables enteras se guardan en dos bytes como números en complemento a dos. Los números en coma flotante necesitan cinco bytes para guardar la mantisa, un bit de signo y el exponente. Las variables alfanuméricas se guardan en un área diferente de la memoria, tomando un byte por cada carácter de la serie. Sin embargo este dato no se guarda en la tabla; en su lugar, se usan tres bytes para dar el número de caracteres de la serie y la dirección de 16 bits que apunta al inicio del área de datos de la serie

Bien atado
El programa *Rotsub* hace dar vueltas a la figura de líneas sobre la pantalla en alta resolución. Esta figura se define por medio de cuatro tablas, tres de las cuales se emplean para contener las coordenadas X,Y,Z de cada nodo. Una cuarta tabla, E%(,), se emplea para definir cuándo dos nodos determinados deben unirse con una línea. En el ejemplo, las cuatro tablas están inicializadas para producir una figura en forma de cubo



carse por medio de un número de puntos (o nodos) y una matriz de conexiones de los extremos. Los nodos tienen las coordenadas X(I),Y(I),Z(I), donde I vale desde 1 hasta NP (el número de puntos). La matriz de conexiones de esquinas es E%(I,J), donde I y J van desde 1 hasta NP. Si el punto I se conecta con el punto J,E%(I,J) valdrá uno. De lo contrario, será igual a cero. Este procedimiento no es, desde luego, el más económico en memoria, dado que empleamos dos bytes cuando bastaría con uno. Pero se facilita al usuario la tarea de especificar los puntos que han de conectarse. En aplicaciones prácticas, NP tampoco será muy grande.

Para comunicarnos desde el BASIC con la rutina de rotación en código máquina, colocaremos (POKE) las direcciones de base de las tablas comenzando por el elemento 1. Así, las direcciones X(1),Y(1),Z(1) y E%(1,1) no es necesario hallarlas. Además, suministraremos al código máquina el número de puntos, NP, y el coseno (COS) y seno (SIN) del ángulo de rotación deseado alrededor del eje Z.

Todo proyecto ambicioso en código máquina debe proceder por pequeños pasos si no se desea caer en errores desastrosos más adelante. Por ello hemos planificado el proyecto en tres fases. En la primera definimos los algoritmos y escribimos el texto del programa en BASIC. Esto se realiza en el programa *Cubo giratorio en* BASIC (página contigua), que hace dar vueltas a un cubo de tres dimensiones alrededor del eje Z y proyecta el resultado en el plano X,Y.

Nuestro otro objetivo es convertir el listado BASIC en código máquina. Pero comenzamos con una tarea menos ambiciosa consistente en dar la subrutina de la línea 1800 en su versión de código máquina. Esto es lo que hace el programa *I-Rotsub*, dando además un programa de comprobación de esta rutina.

En el próximo capítulo completaremos el proyecto y añadiremos el resto de la codificación al listado assembly, así como daremos un "Cargador en BASIC" del programa completo.

Programa I-Rotsub 64

El siguiente listado en assembly será ensamblado y cargado y el código resultante cargado como I-ROT.HEX

```
; ++++++++++++++++++++++
;++          I-ROTSUB 64          ++
; ++++++++++++++++++++++
;++    EMPLEA TABLAS DEFINIDAS    ++
;++ DESDE BASIC SE DEBEN COLOCAR ++
;++    LAS DIRECCIONES PRIMERO    ++
; ++++++++++++++++++++++

;PLTSUB   = $C183
;XLO      = $C103
;XHI      = $C104
;YLO      = $C105
;
*         = $C544

;++++ VARIABLES ROTSUB ++++

; VARIABLES LLAMADAS DESDE BASIC
XBASLO    *=*+1 ; POKE50500,X(0)LO
XBASHI    *=*+1 ; POKE50501,X(0)HI
XBASLO    *=*+1 ; POKE50502,Y(0)LO
YBASHI    *=*+1 ; POKE50503,Y(0)HI
NP        *=*+1 ; POKE505D4,NP
; VARIABLES EMPLEADAS POR C/M
XILO      *=*+1
XIHI      *=*+1
YILO      *=*+1
YIHI      *=*+1
CSLO      *=*+1
CSHI      *=*+1
SNLO      *=*+1
SNHI      *=*+1
MEM1      *=*+5 ; VAR COMA FLOTANTE
MEM2      *=*+5 ; VAR COMA FLOTANTE
FAC       =$0061
ARG       =$0069

; LLAMADAS ARITMETICAS AL INTERPRETE
;
FMULT     =$BA28   ; FAC=FAC*ARG
FADDT     =$B86A   ; FAC=FAC+ARG
FSUB      =$B850   ; FAC=MEM-FAC
FADD      =$B867   ; FAC=FAC+MEM
MOVFM     =$BBA2   ; FAC=MEMORY
MOVMF     =$BBD4   ; MEMORY=FAC
;
;++++ GUARDA LOS REGISTROS
;
          PHA
          TXA
          PHA
          TYA
          PHA
;
;++++ INICIALIZA VARIABLES ++++
;
          LDA  XBASLO
          STA  XILO
          LDA  XBASHI
          STA  XIHI
          LDA  YBASLO
          STA  YILO
          LDA  YBASHI
          STA  YIHI
;
;++++ REALIZA MEM1=X(I)*CS-Y(I)*SN
;
START
          LDA  XILO
          LDY  XIHI
          JSR  MOVFM    ; FAC=X(I)
          LDA  CSLO
          LDY  CSHI
          JSR  FMULT    ; FAC=X(I)*CS
          LDX  #<MEM1
          LDY  #>MEM1
          JSR  MOVMF    ; MEM1=X(I)*CS
          LDA  YILO
          LDY  YIHI
          JSR  MOVFM    ; FAC=Y(I)
          LDA  SNLO
          LDY  SNHI
          JSR  FMULT    ; FAC=Y(I)*SN
          LDA  #<MEM1
          LDY  #>MEM1
          JSR  FSUB     ; FAC=MEM1-FAC
          LDX  #<MEM1
          LDY  #>MEM1
          JSR  MOVMF    ; MEM1=FAC
;
;++++ REALIZA MEM2=Y(I)*CN+X(I)*SN
;
          LDA  YILO
          LDY  YIHI
          JSR  MOVFM    ; FAC=Y(I)
          LDA  CSLO
          LDY  CSHI
          JSR  FMULT    ; FAC=Y(I)*CS
          LDX  #<MEM2
          LDY  #>MEM2
          JSR  MOVMF    ; MEM2=Y(I)*CS
          LDA  XILO
          LDY  XIHI
          JSR  MOVFM    ; FAC=X(I)
          LDA  SNLO
          LDY  SNHI
          JSR  FMULT    ; FAC=X(I)*SN
          LDA  #<MEM2
          LDY  #>MEM2
          JSR  FADD     ; FAC=MEM2+FAC
          LDX  #<MEM2
          LDY  #>MEM2
          JSR  MOVMF    ; MEM2=FAC
;
;++++ REALIZA X(I)=MEM1:Y(I)=MEM2
;
          LDA  #<MEM1
          LDY  #>MEM1
          JSR  MOVFM    ; FAC=MEM1
          LDX  XILO
          LDY  XIHI
          JSR  MOVMF    ; X(I)=FAC
          LDA  #<MEM2
          LDY  #>MEM2
          JSR  MOVFM    ; FAC=MEM2
          LDX  YILO
          LDY  YIHI
          JSR  MOVMF    ; Y(I)=FAC
;
;++++ BUCLE COMPROBACION FIN
;
          DEC  NP
          BEQ  EXIT
;
;++++ INCREMENTA PUNTEROS TABLA
;
          LDA  #$05
          CLC
          ADC  XILO
          STA  XILO
          BCC  XNOHI
          INC  XIHI
XNOHI
          LDA  #$05
          CLC
          ADC  YILO
          STA  YILO
          BCC  YNOHI
          INC  YIHI
YNOHI
          JMP  START
;
;++++ SACA REGISTROS DE LA PILA ++++
;
EXIT
          PLA
          TAY
          PLA
          TAX
          PLA
;
          RTS
.END
```

Cubo giratorio en BASIC

```
1000 REM**CUBO GIRATORIO EN BASIC**
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 REM**DIMENSION TABLAS**
1040 NP=8:REM NUMERO DE PUNTOS
1050 DIM X(NP),Y(NP),Z(NP)
1060 DIM ED(NP,NP):REM CONEXION ESQUINAS
1070 REM**INICIALIZACION TABLAS
1080 REM--DATOS COORDENADAS CUBO
1090 DATA  75,  75,  75:REM  --------- /1
1100 DATA -75,  75,  75:REM     CUATRO PTS  /2
1110 DATA -75,-75,  75:REM     SUPERIORES  /3
1120 DATA  75,-75,  75:REM  --------- /4
1130 DATA  75,  75, -75:REM  --------- /5
1140 DATA -75,  75, -75:REM     CUATRO PTS  /6
1150 DATA -75,-75, -75:REM     INFERIORES  /7
1160 DATA  75,-75, -75:REM  --------- /8
1170 REM**CUBO GIRA PI/4 SOBRE EJE *
1180 FORI=1 TONP
1190 READX(I),Y(I),Z(I)
1200 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1210 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1220 NEXT
1230 REM**CUBO GIRA PI/4 SOBRE EJE Z
1240 FORI=1 TONP
1250 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1260 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1270 NEXT
1280 REM--DATOS CONEXION ESQUINAS--
1290 E(1,2)=1:REM 1 CONECTA CON 2
1300 E(2,3)=1:E(3,4)=1:E(4,1)=1
1310 E(5,6)=1:REM CUADRO DE BASE
1320 E(6,7)=1:E(7,8)=1:E(8,5)=5
1330 E(5,1)=1:REM ESQUINAS DE ARRIBA ABAJO
1340 E(6,2)=1:E(7,3)=1:E(8,4)=1
1350 REM**SIMETRIA E(I,J)**
1360 FORI=1 TONP:FORJ=1 TONP
1370 IFE(I,J)=1 THENE(J,I)=1
1380 NEXT:NEXT
1390 REM######################
1400 REM##TRAZADO CUBO GIRATORIO##
1410 SA=2*π/45
1420 FOR A=π/4 TO π/4+2*π STEP SA
1430 GOSUB1800:REM GIRA POR SA
1440 GOSUB1590:REM INIC/BORR PANTALLA
1450 REM--TRAZADO CUBO--
1460 FORI=1 TONP
1470 FORJ=1 TOI
1480 IF E(I,J)=0THEN1510:REM NO UNIDO
1490 GOSUB1630:REM CALCULA PROYECCION
1500 GOSUB1670:REM UNE PUNTOS
1510 NEXT:NEXT
1520 REM---------------
1530 NEXT A:REM ANGULO SIGUIENTE
1540 REM######################
1550 REM**ESPERA**
1560 GETA$:IFA$=""THEN1560
1570 GOSUB1760:REM RESTAURA PANTALLA
1580 END
1590 REM**ESTABLECE ALT. RES**
1600 POKE49408,1:POKE49409,1
1610 POKE49410,1:SYS49422
1620 RETURN
1630 REM**CALCULA PROYECCION EN ALT. RES.**
1640 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1650 X2%=X(J)+159:Y2%=199-(ZJ)+100)
1660 RETURN
1670 REM**LINESUB**
1680 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1690 MHI=INT(X1%256):MLO=X1%-256*MHI
1700 NHI=INT(X2%i256):NLO=X2%-256*NHI
1710 POKE49920,MLO:POKE49921,MHI
1720 POKE49922,NLO:POKE 49923,NHI
1730 POKE49924,Y1%;POKE49925,Y2%
1740 SYS49934:REM LINESUB
1750 RETURN
1760 REM**RESTAURA PANTALLA**
1770 POKE49408,0:SYS49422
1780 PRINTCHR$(147)
1790 RETURN
1800 REM**CUBO GIRA SOBRE EJE Z/SA
1810 FORI=1 TONP
1820 X(I)=X(I)*COS(SA)-Y(I)*SIN(SA)
1830 Y(I)=Y(I)*COS(SA)+X(I)*SIN(SA)
1840 NEXT
1850 RETURN
```

Programa prueba I-Rotsub

Debe entrarse y guardarse con el nombre PRUEBA I-ROT. No pueden definirse variables mientras se ejecuta el código entre 1880 y 2000 y se llama al SYS50523. De lo contrario la dirección de base de las tablas se pasaría incorrectamente al cód. máq. y el programa se arruinaría

```
1000 REM** PRUEBA I-ROT **
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"I-ROT.HEX",8,1
1040 REM**DIMENSION TABLAS**
1050 NP=8:REM NUMERO DE PUNTOS
1060 DIM X(NP),Y(N),Z(NP)
1070 DIM ED(NP,NP):REM CONEXION ESQUINAS
1080 REM**INICIALIZACION TABLAS**
1090 REM--DATOS COORDENADAS CUBO
1100 DATA  75,  75,  75:REM  --------- /1
1110 DATA -75,  75,  75:REM     CUATRO PTS  /2
1120 DATA -75, -75,  75:REM     SUPERIORES  /3
1130 DATA  75, -75,  75:REM  --------- /4
1140 DATA  75,  75, -75:REM  --------- /5
1150 DATA -75,  75, -75:REM     CUATRO PTS  /6
1160 DATA -75, -75, -75:REM     INFERIORES  /7
1170 DATA  75, -75, -75:REM  --------- /8
1180 REM**GIRO SOBRE EJE X EN PI/4
1190 FORI=1 TONP
1200 READX(I),Y(I),Z(I)
1210 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1220 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1230 NEXT
1240 REM**GIRO SOBRE EJE Z EN PI/4
1250 FORI=1 TONP
1260 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1270 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1280 NEXT
1290 REM--DATOS CONEXION ESQUINAS--
1300 E(1,2)=1:REM 1 CONECTADO A 2
1310 E(2,3)=1:E(3,4)=1:E(4,1)=1
1320 E(5,6)=1:REM CUADRADO INFERIOR
1330 E(6,7)=1:E(7,8)=1:E(8,5)=5
1340 E(5,1)=1:REM ESQUINAS DE ARRIBA ABAJO
1350 E(6,2)=1:E(7,3)=1:E(8,4)=1
1360 REM**SIMETRIA E(I,J)**
1370 FORI=1 TONP:FORJ=1 TONP
1380 IFE(I,J)=1THENE(J,I)=1
1390 NEXT:NEXT
1400 REM######################
1410 REM##TRAZADO CUBO GIRATORIO##
1420 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1430 FOR A=0 TO 2*π STEP SA
1440 GOSUB1570:REM GIRO POR SA
1450 GOSUB1600:REM INIC/BORRADO PANTALLA
1460 REM--TRAZADO CUBO--
1470 FORI=1 TONP
1480 FORJ=1TOI
1490 IF E(I,J)=0THEN1520:REM NO UNIDO
1500 GOSUB1640:REM CALCULO PROYECCION
1510 GOSUB1680:REM UNION PUNTOS
1520 NEXT:NEXT
1530 REM---------------
1540 NEXT A:REM ANGULO SIGUIENTE
1550 REM######################
1560 REM**ESPERA**
1570 GETA$:IFA$=""THEN1570
1580 GOSUB1770:REM RESTAURA PANTALLA
1590 ENU
1600 REM**ESTABLECE ALT. RES.**
1610 POKE49408,1:POKE49409,1
1620 POKE49410,1:SYS49422
1630 RETURN
1640 REM**CALCULA PROYECCION EN ALT. RES.**
1650 X1%=X(I)+159:Y1%=199-(Z(I)+100)
1660 X2%=X(J)+159:Y2%=199-(Z(J)+100)
1670 RETURN
1680 REM**LINESUB**
1690 IF(X1%=X2%)AND(Y1%=Y2%)THENRETURN
1700 MHI=INT(X1%/256):MLO=X1%-256*MHI
1710 NHI=INT(X2%256):NLO=X2%-256*NHI
1720 POKE49920,MLO:POKE49921,MHI
1730 POKE49922,NLO:POKE49923,NHI
1740 POKE49924,Y1%:POKE49925,Y2%
1750 SYS49934:REM LINESUB
1760 RETURN
1770 REM**RESTABLECE PANTALLA**
1780 POKE49408,0:SYS49422
1790 PRINTCHR$(147)
1800 RETURN
1810 REM**GIRO CUBO SOBRE EJE Z/SA
1820 FORI=1 TONP
1830 X(I)=X(I)*CS-Y(I)*SN
1840 Y(I)=Y(I)*CS+X(I)*SN
1850 NEXT
1860 RETURN
1870 REM**GIRO SOBRE EJE Z/SA
1880 X(1)=X(1):REM VAR X(1) ACTUAL
1890 POKE50500,PEEK(71):REM BYTE LO DE X(1)
1900 POKE50501,PEEK(72):REM BYTE HI DE X(1)
1910 Y(1)=Y(1):REM VAR Y(1)ACTUAL
1920 POKE50502,PEEK(71):REM BYTE LO DE Y(1)
1930 POKE50503,PEEK(72):REM BYTE HI DE Y(1)
1940 POKE50504,NP:REM NUMERO DE PUNTOS
1950 CS=CS:REM HACE A CS VAR ACTUAL
1960 POKE50509,PEEK(71)
1970 POKE50510,PEEK(72)
1980 SN=SN:REM HACE A SN VAR ACTUAL
1990 POKE50511,PEEK(71)
2000 POKE50512,PEEK(72)
2010 SYS50523
2020 RETURN
```

Luz y sombra

Dimension Graphics

En guardia
La forma más apropiada de definir *Archon* es decir que se trata de un juego de ajedrez animado. Cuando se le pide a una pieza que se desplace a otro cuadrado del tablero, marchará, se escabullirá o volará, según el tipo de personaje que represente. Una vez allí, si la casilla está ocupada por una pieza del bando oponente, la pantalla pasará a un primer plano del cuadrado y comenzará la batalla. El resultado de ésta se decide en función de la destreza del jugador y de la fortaleza relativa de las piezas contrarias

El juego "Archon" combina, con gran originalidad, elementos propios del ajedrez con gráficos de movimiento rápido

Archon es un juego que puede resultarle fascinante al usuario aficionado al ajedrez y que, al mismo tiempo, puede satisfacer al más apasionado entusiasta de los juegos recreativos. La eterna batalla entre fuerzas de la Luz y la Oscuridad constituye el nudo de la trama. El juego comienza con una visualización de un tablero de estrategia en el cual los dos bandos contendientes están dispuestos en filas y columnas, de forma muy similar a las piezas del ajedrez. Las piezas poseen nombre tales como "fénix" y "caballero" en el lado de Luz, mientras que "espíritus", "gnomos" y "dragones" distinguen a las fuerzas de la Oscuridad. Los iconos que representan las piezas de ambos bandos poseen diferentes fuerzas y capacidades de movimiento. Algunas piezas, por ejemplo, pueden "volar", lo que significa que pueden saltar por encima de las piezas que tienen por delante, mientras que otras tienen sólo un movimiento a nivel de "tierra".

El tablero propiamente dicho está dividido en una cuadrícula de nueve por nueve. A primera vista, parece muy similar a un tablero de ajedrez, con muchos de los cuadrados coloreados alternadamente en blanco y negro. Sin embargo, otras casillas cambian de blanco a negro y otra vez a blanco a medida que va transcurriendo el juego. El motivo de ello es que las fuerzas de la Luz son más fuertes en los cuadrados blancos y las fuerzas de la Oscuridad son más fuertes en los negros.

El objetivo de *Archon* consiste en ocupar los cinco "puntos de poder"; cuatro están situados en cruz sobre los bordes del tablero, con el punto crucial en el cuadrado del centro. Cuando comienza el juego, la estrategia de apertura determina que usted mueva sus iconos desde su color opuesto, donde son vulnerables al ataque, a colores en los cuales son más fuertes. Al hacer esto, también puede abrir la fila trasera de modo que las piezas de "tierra" queden libres para moverse, permitiéndole construir una barrera eficaz para impedir que las fuerzas de tierra contrarias desbaraten la defensa.

En este punto, el juego parece tener transferidos al ordenador la mayoría de los elementos del ajedrez. No obstante, la verdadera diferencia del juego se hace evidente al intentar ocupar un cuadrado que ya esté en poder de un icono contrario. Cuando se mueve un icono a un cuadrado ocupado por el enemigo, mediante la palanca de mando, la pantalla revela un primer plano del cuadrado, en vez de capturar automáticamente la pieza. Las piezas del enemigo están posicionadas a los lados del cuadrado, en el cual hay un barra que representa la fuerza del icono, y comienza una contienda estilo recreativo. Cuando la barra es golpeada por la pieza contraria, esta fuerza disminuirá, y cuando desaparezca por completo, el enemigo gana y ocupa el cuadrado. Dado que las fortalezas y los métodos de ataque difieren radicalmente de un icono a otro. Algunas de estas batallas están más niveladas que otras. Por ejemplo, un dragón puede lanzar su llama a través del cuadrado a un contrincante, mientras que un caballero ha de estar muy cerca de un enemigo para poder hacer uso de su espada. Las batallas se complican aún más en función de barreras diseminadas a través del cuadrado, que cambian a medida que transcurre la acción.

Cada bando cuenta con un icono que puede realizar un encantamiento: un hechicero para la Luz y una hechicera para la Oscuridad. Los encantamientos son idénticos, pero el icono sólo los puede utilizar una vez. Los hechizos incluyen poderes tales como "resucitar" una pieza en un cuadrado alejado de su capacidad de movimiento, o invocar a una criatura conocida como un "elemental". A éstas se las puede convocar para que ataquen a una pieza enemiga y son útiles cuando una pieza enemiga amenaza atacar a una de las suyas mucho más débil, o si usted desea debilitar a su oponente. Sin embargo, incluso si el elemental destruye al icono enemigo, al final de la batalla desaparece. Un elemental no puede ocupar un cuadrado.

Archon se puede jugar contra el ordenador o bien contra otra persona. Aunque el ordenador es sólo un estratega mediano, en la sección recreativa resulta ser un oponente formidable, y puede que en las primeras partidas el principiante se encuentre demasiado desbordado para ganar siquiera una sola batalla.

Archon: Para el Commodore 64
Editado por: Ariolasoft, Dept A.S. 72, Westfields Avenue, London, SW13 OAU, Gran Bretaña
Autores: Anne Westfall, Jon Freeman, Paul Reiche III
Palanca de mando: Necesaria
Formato: Cassette

Chris Stevens

Cuatro diseños
Durante los últimos años han salido al mercado numerosas impresoras térmicas. Aunque algunas máquinas, como la Floyd 40 y la Alphacom 32, están destinadas específicamente a su empleo con el Sinclair Spectrum, hay otras máquinas, como la Epson P40 y la Brother HR-5, que están diseñadas para trabajar con una amplia gama de micros personales

Hot metal

Analizaremos cuatro impresoras para el Spectrum y enunciaremos los criterios que deben guiar al usuario al hacer su elección

Uno de los primeros periféricos cuya compra consideran los usuarios de un ordenador personal es la impresora, que les permite producir copias de los listados de sus programas. Al elegir una impresora los usuarios del Spectrum se encuentran con una serie de problemas. Por una parte, este ordenador carece de las interfaces convencionales que por lo general utilizan las impresoras y, por otra, la impresora de Sinclair Research, la ZX Printer, cuya producción se ha suspendido, se caracterizaba por su confusa impresión y la tendencia de las copias a perder nitidez con el paso del tiempo. Debido a que Sinclair Research ha decidido no mejorar su impresora, otros fabricantes han producido por su propia cuenta impresoras térmicas de costo reducido. En este capítulo vamos a examinar algunas de estas alternativas.

A la hora de comprar una impresora se han de tener presentes varias consideraciones básicas. La principal es, por supuesto, el precio, consideración que con frecuencia implica mucho más que limitarse a comparar etiquetas de precio. Puede haber, por ejemplo, costos ocultos que no se incluyen en el precio de la máquina. El fabricante de una impresora puede afirmar que ésta está diseñada para un Spectrum, cuando en realidad requiere un conector de interface RS232C. Por consiguiente, antes de poder utilizar la impresora, también habrá de adquirirse una ampliación Interface 1 (que conecta al Spectrum un conector RS232C).

Costos ocultos

Otro costo oculto es el papel para la impresora. Muchas impresoras sólo utilizan papel diseñado específicamente para ellas. Por ejemplo, las impresoras de tipo térmico como la propia ZX Printer requieren un papel especial sensible al calor. Por lo tanto, tras pagar por su impresora, usted se verá de hecho ligado a los caprichos del fabricante, y el papel puede ser difícil de conseguir o bien tener un precio excesivo.

Al adquirir una impresora es esencial, en consecuencia, asegurarse de que la tienda que vende el dispositivo posea siempre existencias regulares de papel y otros servicios, y que usted sepa cuánto es probable que le cuesten. También ha de recabar información sobre el servicio técnico. La maquinaria que posee partes móviles mecánicas, como el cabezal de impresión, es más susceptible de sufrir desperfectos que los componentes electrónicos y, por tanto, es preferible comprar la impresora en una empresa que posea un eficiente servicio técnico.

Por último hay que considerar el problema de la compatibilidad de software. Todas las impresoras

están programadas para responder a códigos (por lo general, caracteres ASCII) que les indican que han de llevar a cabo ciertas funciones, tales como un retorno de carro, la cantidad de caracteres por línea y la densidad de impresión. Aunque muchos de estos códigos son idénticos a los que utiliza el Spectrum, quizá no sea así para todas las máquinas.

Sinclair Research, como muchos otros fabricantes de micros personales, ha adaptado muchos de los códigos ASCII para su propio uso y, aunque el manual de la impresora indique que un código tiene un determinado significado, bien podría tener otro distinto para el Spectrum. Aquí la solución consiste en ver las facilidades de la impresora operando con este ordenador antes de comprarla.

Las mismas precauciones son válidas para el software que piense utilizar. No hay nada más frustrante que unos códigos de instrucciones que funcionen perfectamente bien desde BASIC pero que no se puedan emplear desde un determinado paquete para tratamiento de textos. En este caso, usted se vería en la nada envidiable situación de tener que guardar su copia en cinta o disco, desconectar y volver a leer la copia como archivo secuencial para poder imprimirla.

Habiendo enumerado varios criterios para la adquisición de una impresora para el Spectrum, examinemos ahora algunas de las impresoras térmicas económicas disponibles en el mercado. De las impresoras que vemos a la derecha, la más barata es la Alphacom 32. Es, asimismo, la máquina que guarda un mayor parecido con la ZX Printer. Las fuentes de tipos utilizadas son como las de ésta y, al igual que en el caso de la máquina de Sinclair, sólo puede imprimir un máximo de 32 columnas.

Por alguna razón, los fabricantes han optado por equipar a la Alphacom con su propia fuente de alimentación. La adición de esta caja extra y los cables correspondientes es innecesaria, en especial porque el bus de ampliación del Spectrum posee una fuente de 9 V, que sería suficiente para operar una impresora térmica.

La estructura de la Floyd 40, de Shiva Marketing, parece más bien endeble. La carcasa se dobla con bastante facilidad y el rodillo que retiene el rollo del papel no es más que un delgado tarugo de madera. A pesar de esto, la máquina constituye una notable mejora respecto a la Alphacom.

La potencia se toma del bus de ampliación del Spectrum y por ello no hay ninguna caja extra que instalar. La Floyd 40 también utiliza papel blanco, con lo cual la impresión en negro resulta mucho más legible. Sin embargo, lo que realmente diferencia a esta máquina de la Alphacom y la ZX Printer es el hecho de que acepta ciertos caracteres de control que pueden alterar la salida a la impresora.

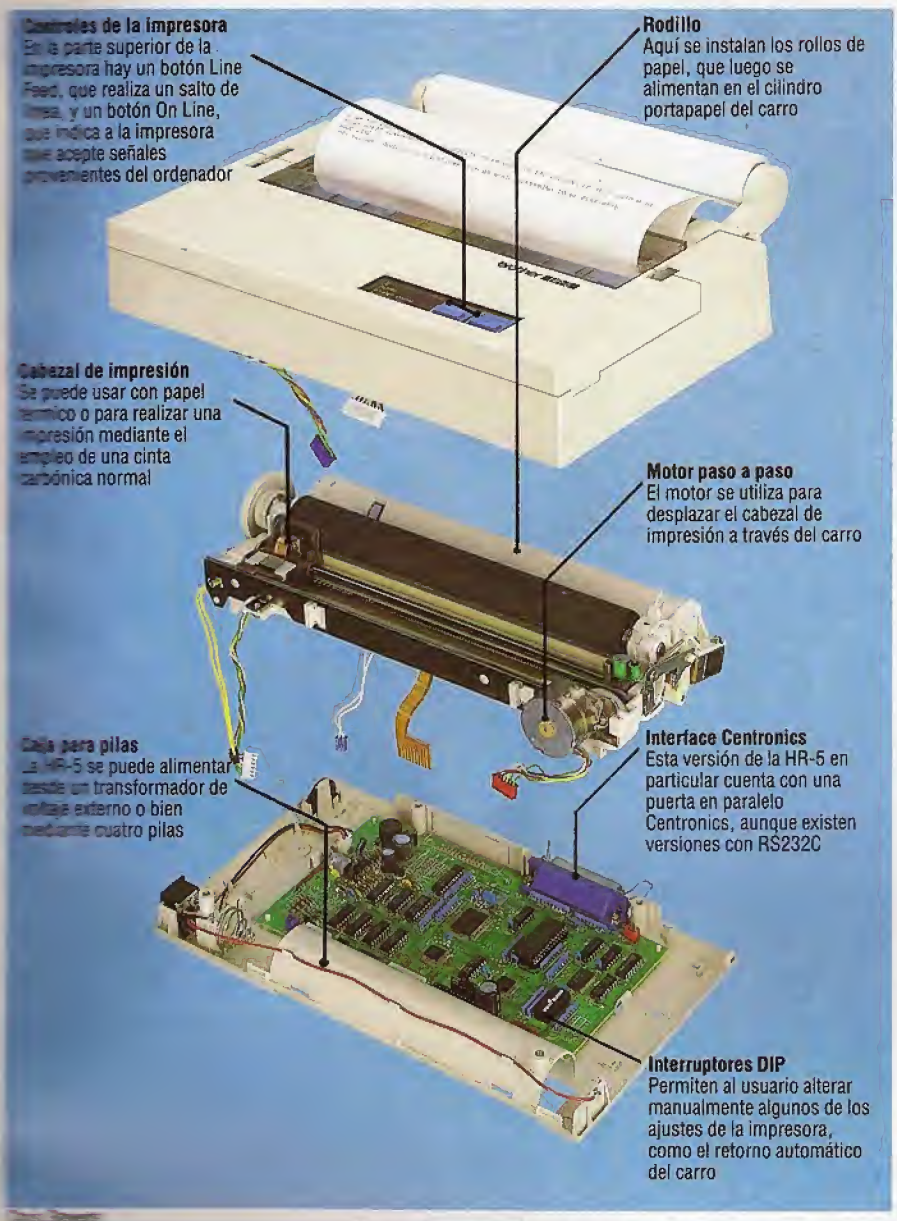
Estos caracteres se envían a la impresora a través de una instrucción LPRINT normal seguida de comillas. El carácter de control se encierra entre signos de exclamación, indicando a la impresora que no lo imprima porque se trata de una instrucción. Por ejemplo, la línea LPRINT"¡H!" le ordenaría a la Floyd 40 que imprimiera en caracteres de doble altura. El formateado se anula impartiendo la misma instrucción una segunda vez.

Otras facilidades disponibles incluyen caracteres para gráficos, cursiva, anchura doble y formato invertido. El cabezal de impresión de cinco por siete agujas produce atractivas imágenes en modalidad

MODELO	Alphacom 32	Floyd 40	Epson P40	Brother HR-5
PRECIO*	£54,95	£69,95	£99,95	£149,95
COSTO PAPEL	£10,95 (3 rollos)	£2,50 (c/rollo;10 rollos,£12,50)	£8,65 (5 rollos)	£4,75 (c/rollo)
INTERFACE EMPLEADA	Bus de ampliación	Bus de ampliación	RS232 o en paralelo Centronics	RS232 o en paralelo Centronics

*En el mercado británico

Interface Kempston
Para poder utilizar algunas de las impresoras más sofisticadas (equipadas con interfaces Centronics o RS232C), el usuario de un Spectrum deberá invertir en un dispositivo periférico adecuado que proporcione estas conexiones. Sin ninguna duda, de las interfaces de este tipo que existen en el mercado la más popular es la interface Kempston, que proporciona acceso a una amplia gama de impresoras compatibles con Centronics



de gráficos y el formato del tipo de impresión es tan satisfactorio como el de muchas otras máquinas que cuestan el doble. Aunque bastante lenta, el único serio inconveniente del sistema es que está limitado a un rollo de papel de 80 mm de ancho.

La tercera máquina que vemos es la Epson P40, que se vende en dos versiones básicas que comprenden los principales tipos de interface para impresora. La P40S es una versión en serie con un conector RS232C, mientras que la P40P posee en la parte posterior una puerta en paralelo Centronics. Sin embargo, puesto que el Spectrum no posee instaladas como estándar ninguna de las dos, es necesario adquirir una Interface 1 o bien una de las otras muchas disponibles en el mercado.

Surge el problema de que el conector RS232 que ofrece Sinclair en la Interface 1 es un conector D de siete patillas no estándar. Los manuales del P40 y la Interface 1 le indicarán de qué forma se han de posicionar las patillas y luego sólo resta una simple labor de soldadura. Sin embargo, si no siente especial inclinación por el uso de soldadores, quizá sea preferible comprar una de las interfaces producidas por empresas independientes de las que hay a la venta. Kempston, por ejemplo, proporciona interfaces que se enchufan en la parte trasera del Spectrum, así como conexiones estándares tanto para dispositivos RS232 como Centronics.

La Epson P40, a pesar de su tamaño relativamente pequeño, puede imprimir tanto en modalidad de 80 como de 40 columnas. Asimismo, la máquina puede hacer uso de una gran cantidad de códigos de escape de los que disponen los dispositivos más grandes de la gama. Por supuesto, el Spectrum no posee una tecla Escape *per se*, de modo que los códigos se deben enviar como códigos ASCII en el formato CHR$(27);"E"; (donde CHR$(27) representa el carácter Escape ASCII) que indica a la P40 que se ponga en modalidad enfatizada.

La P40 también puede llevar a cabo otras numerosas operaciones, tales como alterar el juego de caracteres, entrar en modalidad de imagen de bits (que permite que usted cree sus propios caracteres) y en modalidad condensada. Aparte de estas alteraciones, que se pueden realizar desde software, también hay interruptores DIP, que permiten establecer la paridad y la cantidad estándar de columnas de tipos a imprimir.

El hecho de que la P40 esté amparada bajo la prestigiosa marca comercial Epson hace que uno pueda sentirse bastante seguro de que contará con suficiente apoyo durante algún tiempo.

A diferencia de las otras impresoras que describimos en este capítulo, la Brother HR-5 puede usar papel térmico o bien, de estar equipada con una cinta, papel normal. Otra ventaja de esta máquina respecto a las otras es que el carro puede aceptar papel de tamaño A4 y, por tanto, se la puede utilizar para cartas y otras aplicaciones comunes de tratamiento de textos. La anchura adicional le permite imprimir hasta 132 caracteres por línea.

Al igual que la máquina Epson, la HR-5 hace un uso exhaustivo de códigos Escape para formatear su impresión y, por tanto, puede producir una amplia gama de caracteres, fuentes internacionales y otros formatos. Asimismo, es muy silenciosa en funcionamiento. Otra similitud que comparte la Brother HR-5 con la Epson es que, lamentablemente, la máquina viene con una interface en paralelo Centronics o bien con un conector RS232C. Nuevamente usted habrá, en consecuencia, de adquirir una interface adecuada para poder hacer funcionar la impresora desde un Spectrum. Aparte de está salvedad, la calidad de la salida impresa puede calificarse de excelente.

Al escoger una impresora se plantean problemas sólo si usted decide adquirir una máquina cuyas capacidades estén más allá de las de las máquinas construidas a la medida para el Spectrum. La Alphacom y la Floyd 40 ofrecen la ventaja de que sencillamente se enchufan en el ordenador y que aceptan instrucciones normales del Spectrum, tales como COPY, pero sus capacidades en realidad se limitan a los listados de programas.

Las impresoras situadas en el tramo superior de la escala de precios, con una gama más amplia de ampliaciones, están diseñadas para trabajar con máquinas muy diversas; por consiguiente, no están hechas a la medida para funcionar con el Spectrum. Sin embargo, hasta que alguien produzca una impresora de gran calidad diseñada específicamente para el Spectrum, el usuario habrá, lamentablemente, de adaptar la impresora al ordenador.

Control dinámico

Iniciamos una serie en la que examinaremos detalladamente el sistema operativo CP/M (Control Program for Microprocessors)

Cuando a comienzos de la década de los setenta se inventaron los sistemas de almacenamiento en disco flexible, obviamente existía una apremiante necesidad de desarrollar un sistema operativo que fuera capaz de organizar la información retenida en esos dispositivos sin que los usuarios tuvieran que hacerlo manualmente. Aproximadamente en la misma época, la firma norteamericana de microchips Intel decidió desarrollar un sistema de disco flexible para su serie 8000 de microprocesadores, que había introducido recientemente. El equipo de desarrollo de la empresa está dirigido por un joven ingeniero asesor, Gary Kildall. El programa resultante se dio a conocer como CP/M (originalmente, siglas de *Control Program/Monitor*, que posteriormente se cambiaría por *Control Program for Microprocessors*: programa de control para microprocesadores).

Cuando se lanzó el paquete, en 1975, se convirtió enseguida en un éxito rotundo. De hecho, su aceptación fue tal que cuando un grupo de diseñadores abandonó Intel para formar su propia empresa, Zilog Inc., decidieron que su nuevo chip (el Z80) fuera compatible con el 8008 de modo que pudiera ejectuar CP/M. El propio Kildall también se marchó de Intel para crear su propia empresa, Digital Research, que actualmente es una de las mayores empresas de soffware del mundo.

El éxito del CP/M se puede medir por el hecho de que, aunque durante los años setenta aparecieron muchos sistemas operativos rivales para micros de ocho bits, en especial para aquellos basados en el procesador 6502, no compatible con CP/M, ahora el sistema es el estándar industrial de facto para los micros de gestión de ocho bits. Esto resulta particularmente asombroso si se considera que el CP/M ya tiene más de un decenio de antigüedad, lo que, desde el punto de vista de la microinformática, es un extraordinario período de vida útil.

Para poder ejecutar CP/M se necesita un sistema de ordenador con un microprocesador 8008 o un chip compatible, una unidad de disco flexible y al menos 48 K de memoria disponible. Esta gran cantidad de RAM es necesaria porque con frecuencia se requerirá la memoria para retener el programa CP/M propiamente dicho y cualquiera de los archivos de instrucciones (que pueden ocupar hasta 8 K) y archivos en disco con los cuales puede estar operando. Con el advenimiento de los ordenadores de 16 bits, parece que la época dorada del CP/M en el mercado de gestión ha pasado ya. Pero el sistema ha sido adoptado por muchos fabricantes de ordenadores personales y educativos, posibilitando que sus micros Z80 se aparten del mercado de juegos y pasen a aplicaciones más "serias". La adición de una unidad de disco compatible con CP/M le proporciona instantáneamente al consumidor una inmensa cantidad de software que se desarrolló en los años setenta para micros de gestión basados en el 8008 y el Z80. Ello les permite a los fabricantes obviar el problema del apoyo de software, que ha perjudicado enormemente a muchos micros. Entre los fabricantes que en los últimos años han seguido este camino se incluyen Memotech, Research Machines y Amstrad.

A diferencia de la mayoría de ordenadores personales, que poseen sus sistemas operativos instalados "a bordo" (y que se activan con el encendido), el CP/M casi siempre reside en un disco flexible (denominado *disco de sistema*) que se carga desde una unidad de disco con el encendido. No existe ninguna razón especial por la cual el CP/M haya de estar retenido en un disco de sistema, puesto que algunas empresas han suministrado sistemas similares en ROM. Las causas son básicamente históricas. En los primeros ordenadores, la memoria era difícil de conseguir; de estar el CP/M instalado, cuando no se lo necesitara no hubiera hecho más que ocupar una preciosa memoria que se podría haber utilizado para otros fines. De este modo, se conservó en disco y se lo cargaba (LOAD) sólo cuando era necesario.

Típicamente, un disco de sistema CP/M contiene el programa CP/M propiamente dicho y varios programas más cortos a los que se puede llamar mediante el CP/M para que lleven a cabo funciones específicas de manipulación de disco. El CP/M es, sin embargo, más que un simple sistema de gestión: es un sistema operativo. Cuando se carga el CP/M es un ordenador con su propio sistema operativo residente en ROM, "anula" el sistema de la máquina y asume totalmente la operación del ordenador. Por lo tanto, cuando la máquina está operando bajo CP/M, éste no reconoce las instrucciones en BASIC que normalmente ejectua el ordenador, generando un mensaje de error. De hecho, un programa en BASIC no se ejecutará en absoluto bajo el CP/M a menos que el sistema operativo posea un intérprete o compilador de BASIC añadido al sistema.

Kevin Jones

Tras el encendido, por lo general el disco de sistemas CP/M se ejecuta automáticamente y se visualiza (como podemos apreciar) un directorio con algunas de las instrucciones disponibles. Un atento examen del directorio del disco nos dice numerosas cosas sobre la forma en que está estructurado el CP/M. La serie de instrucciones (como todos los archivos en CP/M) consta de dos partes, empezando por el nombre "primario". Cuando se desea cargar cualquier archivo de instrucciones, simplemente se digita esta parte del nombre completo del archivo junto con el nombre del archivo en el cual deseamos operar, y se carga y ejecuta automáticamente.

La segunda parte del nombre de archivo completo (a la derecha del punto) se conoce como la extensión, que indica al CP/M (y al operador) qué clase de archivo es. Dado que todos los archivos de un disco de sistema CP/M son archivos de instrucciones, todos ellos llevan como sufijo la extensión COM. Más adelante en esta serie veremos otras clases de extensión de archivo.

Otra cosa que destaca cuando se examina el directorio del disco es que el programa CP/M propiamente dicho no se lista. A primera vista, entonces, puede parecer que el CP/M es meramente la suma de sus partes y un conjunto de programas de instrucciones más pequeños. Si bien, en gran medida, ésta es la forma en la que está estructurado el CP/M, un examen más atento del directorio revela que muchas instrucciones vitales (como LOAD) están presentes, mientras que faltan otras (p. ej., SAVE). Es evidente que el directorio no nos está contando toda la historia.

Hemos señalado anteriormente que un sistema operativo debe ser, en la medida de lo posible, transparente, dejando en manos del usuario el menor trabajo posible para alimentar el sistema adecuadamente. Cuando cargamos desde el disco de sistemas, el CP/M se carga en la RAM para el usuario y allí permanece mientras el ordenador esté encendido. También se cargan junto con el programa CP/M principal un número de archivos de instrucciones utilizados comúnmente (llamados programas de utilidad) que también residen en RAM. Por consiguiente, para usar estas instrucciones no hay necesidad de cargarlas desde el disco.

Para evitar cualquier confusión sobre qué instrucciones ya están residentes en RAM y cuáles se han de cargar, se ha hecho que estas instrucciones, al igual que el programa CP/M propiamente dicho, sean "invisibles" para el directorio y se afirma que son instrucciones "incorporadas". Se dice que los archivos de instrucciones residentes en disco son instrucciones "transitorias". Una vez ejecutadas, son desechadas por el sistema operativo y si usted desea volver a utilizarlas son cargadas de nuevo.

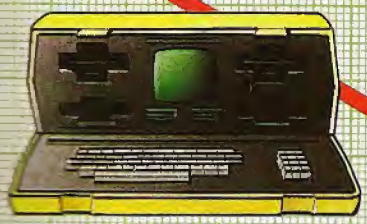
```
WRENMENU version 1.1
(c) 1984 Quantec Systems and Software Ltd
A)dir
A: WRENMENU COM : CPM3     SYS : COPYSYS  COM : ERASE   COM : SET     COM
A: SHOW     COM : SUBMIT   COM : TYPE     COM : DIR     COM : GET     COM
A: PUT      COM : RENAME   COM : SETDEF   COM : PIP     COM : HELP    COM
A: HELP     HLP : FC       COM : DEVICE   COM : FORMAT  SUB : FC      GET
A: FORMAT   GET : CONFIGUR COM
A>
```

Consulta al directorio
Digitando DIR se visualiza una lista de los archivos retenidos en el sistema CP/M. A menos que usted ya haya añadido algunos archivos en el disco, todas las entradas serán archivos de instrucciones (COMmand), identificables en función del sufijo .COM del final de cada nombre de archivo. El directorio visualiza, asimismo, otras informaciones, como la cantidad de memoria que ocupa cada archivo y el nombre del archivo desde el cual se accedió al directorio

Este sistema aparentemente arbitrario de ejecutar las instrucciones se debe, en parte, a limitaciones de hardware, y en parte también a que ello le confiere flexibilidad al sistema. Puesto que el CP/M se basa en procesadores de ocho bits, la máxima memoria que el procesador puede direccionar directamente son 64 K. Si en el encendido se cargaran en la memoria todas las instrucciones necesarias para ejecutar CP/M, quedaría muy poca RAM para programas del usuario y archivos de texto. Por consiguiente, se decidió que para optimizar el uso de la memoria central, ésta sólo contendría las instrucciones más habituales, mientras que el resto se cargarían al ser requeridas y se desecharían luego de ser utilizadas.

Sin embargo, tal sistema posee sus ventajas. Significa que se pueden añadir con bastante facilidad instrucciones transitorias extras con sólo añadirlas al directorio con una extensión .COM. Esta flexibilidad en cuanto a la posibilidad de añadir instrucciones extras que se puedan ejecutar bajo la cobertura general del CP/M ha tenido como consecuencia la inmensa cantidad de programas de aplicaciones que se han escrito para ejecutar bajo CP/M. A modo de ejemplo, el CP/M considera al popular paquete para tratamiento de textos *WordStar* (aunque se lo trata como un programa separado que se puede ejecutar bajo CP/M) como otro de sus archivos de instrucciones cuando el mismo se carga en el sistema. Esto reporta varias ventajas. Para empezar, el *WordStar* se puede usar en cualquier sistema CP/M: el programa no requiere rutinas propias de manipulación de disco, puesto que puede, simplemente, llamar a las instrucciones del CP/M que llevan a cabo estas funciones.

Osborne 1
£1 695



Wren
£1 000

(Los precios que aparecen en estas páginas corresponden al mercado británico.)

Amstrad CPC 664
£339

1981 1982 1983 1984 1985

Provechoso intercambio

Ya en tierra, puede iniciar una interesante actividad de intercambio que, al regresar a casa, le reportará saneados beneficios

La línea 892 de la sección principal del programa llama a la rutina de intercambio, cuya primera tarea consiste en comprobar si usted ha traído algunas armas para comerciar. Aunque las armas pueden ser útiles (para defenderse de los piratas, etc.), al jefe de los nativos no le agradan las armas, porque sabe que éstas podrían ser fuente de problemas entre su pueblo. Esto significa, lamentablemente, que si *ha traído* armas para comerciar con ellas, el jefe rechazará la oferta y se visualizará un mensaje en ese sentido. No obstante, el resto de las mercancías se comerciará en lotes.

La rutina continúa y le informa cuál era el precio en su país, antes de que usted se hiciera a la mar, para cada una de las mercancías que ofrece el jefe. Primero se comercia la sal y usted puede intercambiarla por perlas, figurillas o especias. Tras comerciar con la sal, puede continuar con la tela de la misma manera, y luego con los cuchillos y las joyas.

Si no hay armas en el barco, no será necesario que el jefe le diga al capitán que no quiere armas, de modo que la línea 10072 examina el segundo elemento de la matriz de provisiones, OA(2), que registra la cantidad de armas, e ignora el mensaje del jefe si este elemento está establecido en cero.

La línea 10080 examina los cuatro últimos elementos de la matriz de provisiones (de OA(3) a OA(6)) en busca de mercancías aptas para el comercio. Los dos primeros elementos representan medicinas y armas, que no son aptas y, por consiguiente, no se verifican. En caso de que quedaran cuchillos, sal, tela o joyas, se le dirá que el jefe está deseoso de ofrecer a cambio perlas, figurillas y especias. Pero si todos los elementos de la matriz estuvieran establecidos en cero, el juego terminaría porque, como es obvio, no habría ninguna mercancía con la cual comerciar.

Tras informarle sobre el valor de cada una de las mercancías cuando el barco zarpó, y advirtiéndole que desde entonces los precios podrían haber experimentado variaciones, entre las líneas 10130 y 10200 el programa entra en un bucle para tratar la mecánica del intercambio. El código de este bucle calcula las cantidades de perlas, figurillas y especias ofrecidas por el jefe para cada lote de mercancías que usted desea comerciar, y le pregunta qué mercancías desea. Luego almacena las mercancías elegidas en el barco.

El bucle cuenta de 3 a 6, omitiendo, por consiguiente, las armas y los frascos de medicina (que están retenidos en el primer y segundo elemento de OA()) y se ocupa de cada artículo por separado. La línea 10135 comprueba si el valor del elemento de la matriz que se esté examinando en cada momento es 0, lo que significa que el jugador no posee ninguna mercancía de este tipo y, de ser así, pasa al siguiente elemento de OA() saltando a la instrucción NEXT de la línea 10200. En caso contrario, se informará al jugador la cantidad de cada mercancía, que se imprime mediante la línea 10145. Las líneas 10150-10153 imprimen el tipo de mercancía, correspondiente al valor actual de T.

Después de que el jefe ofrezca algunas perlas, figurillas y especias, el programa calcula las cantidades que recibirá usted de estos artículos por cada una de las mercancías que ofrezca. Para este cálculo se utiliza la matriz bidimensional EQ(,) creada en la línea 63 y que contiene las proporciones de trueque.

La expresión de la línea 10165 calcula la cantidad de perlas ofrecidas para cada elemento del bucle,

multiplicando la cantidad de sus mercancías a intercambiar, OA(T), por la proporción de trueque establecida en la matriz EQ(,). Puesto que el bucle está establecido de 3 a 6, se le debe restar 2 a T para equipararlo con la cantidad de elementos de las provisiones de EQ(,), que son de 1 a 4. Si el bucle se está ocupando de la sal, el contador del bucle T será igual a 3; EQ(T−2,1) corresponde, por consiguiente, a la intersección entre el primer elemento del primer subíndice, sal, con el primer elemento del segundo subíndice, perlas. En EQ(,), éste está establecido en .5; por lo tanto, para la sal, la línea 10165 multiplicará la cantidad de sal por .5 para calcular la cantidad de perlas que se ofrece para el trueque.

La línea 10166 lleva a cabo una función similar para la cantidad de figurillas que ofrece el jefe a cambio. La proporción de trueque para éstas está registrada en el segundo elemento del segundo subíndice de la matriz EQ(,). Cada valor sucesivo del contador del bucle T dirige el programa, por consiguiente, a la sección adecuada de la segunda columna de la matriz. Del mismo modo, la línea 10167 determina la cantidad de especias que ofrece el jefe por cada una de las mercancías de usted, utilizando la tercera columna de la matriz de proporciones de trueque.

Tras informársele de lo que ha ofrecido el jefe, usted debe decidir qué mercancías desea comerciar. Se debe entrar 1, 2 o 3 para el tipo de mercancías requerido, y la línea 10176 determina si la entrada fue correcta asegurándose de que la entrada esté comprendida en la escala de 1 a 3. Las mercancías intercambiadas se deben luego transportar hasta el barco; la línea 10180 almacena la cantidad en la matriz AO(), DIMensionada en la línea 68.

Los tres elementos de esta matriz representan las tres mercancías que tiene el jefe para ofrecer. La cantidad de artículos transportados al barco se vuelve a calcular utilizando la matriz de proporciones de trueque EQ(,), multiplicando el elemento en cuestión por la cantidad de mercancías comerciadas, OA(T). I, el número que usted debe digitar para seleccionar la categoría de mercancías deseada, se utiliza para seleccionar el elemento correcto de AO(), en donde se han de almacenar las perlas, figurillas y especias obtenidas. Asimismo, se emplea I como el segundo subíndice de EQ(,).

Observe que si en un posterior intercambio se obtiene una segunda cantidad de perlas, figurillas o especias, la cantidad se debe sumar a la cantidad ya almacenada en la matriz AO(). Tras registrar en AO() los bienes recientemente obtenidos, el bucle retorna, de ser necesario, al principio. Tras haber intercambiado todas las mercancías termina la actividad comercial y se le informa respecto a las cantidades de perlas, figurillas y especias que se han adquirido y cargado a bordo de su barco. Las líneas 10220 y 10244 imprimen la cantidad de cada artículo, almacenadas ahora en la matriz AO(). El control retorna entonces al programa principal.

En el próximo capítulo concluiremos el juego mercantil *Nuevo Mundo* con la adición de las secciones finales, en las que usted se verá implicado en una insurrección local antes de que emprenda la dilatada travesía de regreso al Viejo Continente y venda las mercancías recientemente adquiridas, esperando obtener suficientes beneficios como para justificar tan largo viaje y los avatares pasados durante la expedición.

Módulo 12: Intercambio

Adición al cuerpo principal del programa

```
892 GOSUB 10070
```

Rutina de intercambio

```
10070 PRINT CHR$(147):GOSUB 9200:REM INTERCAMBIO
10072 IF OA(2)=0THEN 10080
10074 S$="EL JEFE NO QUIERE TUS ARMAS*":GOSUB 9100
10076 S$="PORQUE PODRIAN ACARREARLE PROBLEMAS*":GOSUB
      9100
10078 PRINT:GOSUB 9200
10080 IF OA(3)<>0 OR OA(4)<>0 OR OA(5)<>0 OR OA(6)<>0 THEN
      10100
10085 S$="NO TE QUEDA NINGUNA MERCANCIA*":GOSUB 9100
10090 S$="CON LA QUE COMERCIAR*":GOSUB 9100
10095 GOTO 10038
10100 S$="EN TRUEQUE POR LOS CUCHILLOS*":GOSUB 9100
10102 S$="SAL TELA O JOYAS QUE POSEAS*":GOSUB 9100
10104 S$="TE OFRECE PERLAS, FIGURILLAS*":GOSUB 9100
10106 S$="Y ESPECIAS*":GOSUB 9100
10108 PRINT:GOSUB 9200
10110 S$="CUANDO SALISTE DE PUERTO ESTAS*":GOSUB 9100
10112 S$="VALIAN:*":GOSUB 9100
10114 S$="PERLAS−2 P DE ORO CADA UNA*":GOSUB 9100
10116 S$="FIGURILLAS−2 P DE ORO CADA UNA*":GOSUB 9100
10118 S$="ESPECIAS−1 P DE ORO EL GRAMO*":GOSUB 9100
10120 PRINT:GOSUB 9200
10122 S$="PERO CUANDO VUELVAS A CASA*":GOSUB 9100
10124 S$="QUIZA ESTOS VALORES HAYAN CAMBIADO*":GOSUB
      9100
10125 PRINT:GOSUB 9200:S$=K$:GOSUB 9100
10126 GET I$:IF I$="" THEN 10126
10130 FOR T=3 TO 6
10135 IF OA(T)=0 THEN 10200
10140 PRINT CHR$(147):GOSUB 9200
10145 PRINT"TIENES";OA(T);
10150 IF T=3 THEN S$="SACOS DE SAL*"
10151 IF T=4 THEN S$="BALAS DE TELA*"
10152 IF T=5 THEN S$="CUCHILLOS*"
10153 IF T=6 THEN S$="JOYAS*"
10155 GOSUB 9100
10156 PRINT:GOSUB 9200
10160 S$="A CAMBIO EL JEFE TE OFRECE*":GOSUB 9100
10165 PRINT "YA SEA";OA(T)*EQ(T−2,1);"PERLAS"
10166 PRINT "O BIEN";OA(T)*EQ(T−2,2);"FIGURILLAS"
10167 PRINT"      O";OA(T)*EQ(T−2,3);"GRAMOS DE ESPECIAS"
10168 PRINT:GOSUB 9200
10170 S$="QUIERES PERLAS, FIGURILLAS*":GOSUB 9100
10172 S$="O ESPECIAS?*":GOSUB 9100
10174 S$="(ENTRA 1,2 o 3)*":GOSUB 9100
10175 INPUT I$
10176 I=VAL(I$):IF I<1 OR I>3 THEN 10174
10180 AO(I)=AO(I)+(OA(T)*EQ(T−2,I))
10190 PRINT:PRINT"LAS ";T$(I);" SE CARGAN EN EL BARCO"
10192 S$=K$:GOSUB 9100
10194 GET I$:IF I$="" THEN 10194
10200 NEXT
10210 PRINT:PRINT:GOSUB 9200
10215 S$="      FIN DEL INTERCAMBIO*":GOSUB 9100
10216 PRINT:GOSUB 9200
10218 S$="HAS OBTENIDO:*":GOSUB 9100
10220 PRINT AO(1);"PERLAS"
10222 PRINT AO(2);"FIGURILLAS"
10224 PRINT AO(3);"GRAMOS DE ESPECIAS"
10226 PRINT:GOSUB 9200
10228 S$=K$:GOSUB 9100
10229 GET I$:IF I$="" THEN 10229
10230 RETURN
```

Complementos al BASIC

Spectrum:
Reemplazar EQ(,) por Q(,) y AO() por E() en todo el listado e introducir estos cambios:

```
10070 CLS
10126 LET I$=INKEY$:IF I$="" THEN GO TO 10126
10140 CLS:GO SUB 9200
10229 LET I$=INKEY$:IF I$="" THEN GO TO 10299
```

BBC Micro:
Introducir las siguientes modificaciones:

```
10070 CLS
10126 I$=GET$
10140 CLS:GOSUB 9200
10229 I$=GET$
```

Wattleworth Silk McConachie en asociación con Stu

Gran maestro
La mayoría de los programas inteligentes de ajedrez poseen la capacidad de "anticipar" una cantidad de movimientos en el juego para evaluar cuál de los siguientes movimientos posibles es el mejor. Los ordenadores hacen esto construyendo un árbol de juego y efectuando una búsqueda en él. En los primeros días de la investigación en materia de AI, se consideraba que la capacidad de un ordenador para jugar al ajedrez era la máxima medida de la inteligencia de la máquina

Alta estrategia

Profundicemos en la mecánica de programas capaces de determinar los mejores movimientos en juegos de estrategia

La frase "juego por ordenador" suele traer a la mente una imagen de alienígenas a los que hay que capturar en la inmensidad del espacio o intentar atraer en cavernas subterráneas; pero no siempre es así. En los primeros días de la informática, algunos pioneros (entre ellos celebridades tales como Claude Shannon, John von Neumann y Alan Turing) dedicaron sus esfuerzos a programar un ordenador para que jugara al ajedrez.

El ajedrez se consideraba como el juego intelectual por excelencia, y un programa que jugara al ajedrez con éxito representaba la máxima prueba de la inteligencia de una máquina. En la actualidad hay sistemas de ordenador, como el *Belle* y el *Cray Blitz*, que juegan al nivel de los maestros internacionales, aunque pocas personas afirmarían que estas máquinas piensan. Aun así, el ajedrez y otros juegos de destreza mental ofrecen un campo ideal para poner a prueba teorías de planificación estratégica: la competición abierta.

La mayor parte de los programas de juegos de destreza se basan en técnicas de búsqueda arborescente, bastante similares a las descritas en el capítulo anterior, pero con modificaciones para tener en cuenta a un adversario. La idea fundamental es la de la "anticipación". El programa construye un árbol de juego considerando sus propios movimientos, analizando los contramovimientos que tiene a su disposición el oponente, anticipando sus respuestas a los mismos y así sucesivamente.

El diagrama del árbol de juego muestra el árbol anticipado para un juego imaginario entre dos personas. La raíz del árbol es la posición actual, con MAX listo para mover. Los nudos terminales, u hojas, son posiciones fin-del-juego. El árbol se utiliza para seleccionar el movimiento a efectuar mediante un procedimiento denominado *minimaximización*, que fue enunciado claramente por primera vez en 1949 por Claude Shannon. Trabaja asignando primero valores numéricos a los nudos terminales: supongamos uno para ganado, cero para empate y uno negativo para perdido. Estos valores se combinan luego al ir recorriendo el árbol, sobre el supuesto de que el jugador (MAX) siempre recoge los mayores mientras que el oponente (MIN) siempre elige los menores, para producir valores para nudos más altos.

En este ejemplo el valor de la raíz es 0, indicando que el juego producirá un empate (siempre y cuando ninguna de las partes cometa ningún error). El movimiento correcto en el nivel superior es, por tanto, M1, M3 o M4, pero no M2. Las reglas que rigen la bifurcación y generación de valores de nudos están determinadas por las reglas del juego específico. Sólo en los juegos triviales, como en el tres en raya, se puede construir todo el árbol del juego completo hasta el final. El ajedrez, por ejemplo, posee un *factor de ramificación* de alrededor de 32, lo que significa que en cualquier posición hay aproximadamente 32 movimientos legales. La anticipación de cuatro niveles (dos movimientos por cada lado) conduciría a más de un millón de nudos terminales. Esta "explosión combinatoria" significa que los programas para jugar al ajedrez no pueden anticiparse hasta la conclusión del juego.

En cambio, la mayoría de los programas de juegos se anticipan hasta donde pueden y evalúan las posiciones allí halladas. Para hacerlo se requiere un método para juzgar cuán favorables o desfavorables son los nudos de las hojas, aun cuando no se sepa el verdadero resultado. Ésta se suele denominar *función de evaluación estadística* e introduce necesariamente imprecisión, porque sólo es una estimación del resultado final. No obstante, la razón de ser de analizar una cierta distancia de anticipación y utilizar una función de evaluación imperfecta, es que se aplicará cerca del final del juego y será, con toda probabilidad, una estimación más acertada que la efectuada sin realizar búsqueda alguna.

Tomando a modo de ejemplo el juego de las damas, podríamos elaborar una función de evaluación muy simple de cuatro términos basada en:

D Ventaja de dama
P Ventaja de pieza

M Diferencia de movilidad
C Control del centro

Estos atributos se pueden calcular examinando el tablero. Por ejemplo, D=WD−BD, donde WD es la cantidad de damas defensoras y BD representa la cantidad de damas oponentes.

Las otras variables reflejan varios hechos: es mejor tener más piezas que el oponente (el perdedor acaba sin ninguna pieza); es útil disponer de mayor movilidad, y los cuadrados centrales, en las damas así como en el ajedrez, son más valiosos que los cuadrados laterales. El programa debe combinar de alguna forma estas cantidades en un marcador global.

Suponiendo que decidimos que una dama (D) es tres veces más valiosa que una pieza común (P), que una pieza extra vale dos movimientos y medio adicionales (M) y que un movimiento de más es el doble de meritorio que controlar un cuadrado central más, nuestra función de evaluación sería:

V=15D+5P+2M+C

(Típicamente se utilizan estimaciones de enteros por la mayor velocidad de cálculo.)

Ésta, sin embargo, es una función de evaluación muy torpe. A modo de comparación, el clásico programa de Arthur Samuel para jugar a las damas, de principios de los años sesenta, empleaba hasta 25 parámetros. Aquí los coeficientes son, asimismo, bastante arbitrarios. Parte de la diversión al desarrollar programas de juegos reside en ajustar tales estimaciones para obtener un buen equilibrio. Uno de los puntos más exquisitos del programa de Samuel era que regulaba sus propias estimaciones automáticamente, lo que constituía una clase rudimentaria de aprendizaje.

En el transcurso de los últimos treinta años, la idea de otorgar valores numéricos a las características del juego y combinarlos en una suma estimada para calcular el valor de cualquier posición ha demostrado su valor. La función de evaluación juega un papel similar al de la medición de distancia heurística en la solución de problemas mediante la búsqueda, analizada anteriormente.

Un programa que simplemente anticipe una profundidad fijada y evalúe los nudos terminales que se han hallado, se encontrará con problemas. Esto se debe a que algunas posiciones de los juegos son "tranquilas" mientras que otras son sumamente "inestables". En ajedrez es probable que tras una captura o coronación de peón el estado del juego sea muy inestable: en el siguiente movimiento se podría producir una recuperación. Si esto tiene lugar un nivel más allá del "horizonte" del programa, la evaluación será sumamente engañosa.

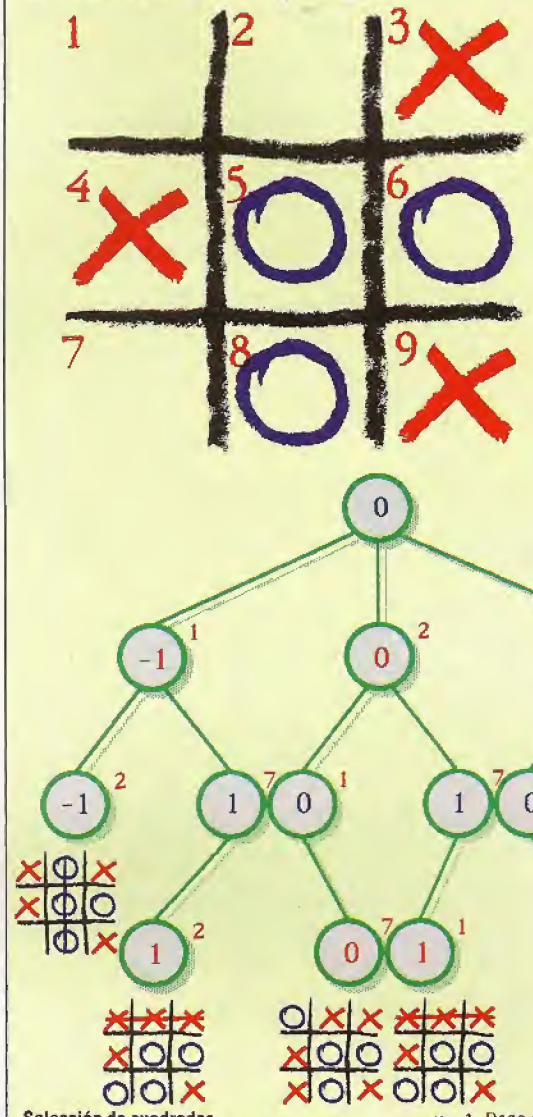
Para reducir este efecto, no poseen una anticipación de profundidad fija. Poseen una medida de "inmovilidad", que indica si una posición se puede evaluar de forma fiable. Si la evaluación no es segura, la búsqueda se empuja un poco más hacia adelante. En ajedrez y damas, esto implica examinar secuencias de capturas comparativamente largas.

El algoritmo *alfa-beta* apareció por primera vez en 1967, en el programa MacHack de Greenblatt. Es un refinamiento de la minimaximización básica y ofrece el mismo resultado pero con muchísimo menos esfuerzo. El diagrama de la página siguiente nuestra parte de un árbol de juego entre dos jugadores llamados MINI y MAX.

Las letras junto a cada nudo (de la A a la L) muestran el orden por el cual se examina el árbol, utilizando un procedimiento de primero en profundidad; los números son evaluaciones. Las barras simples señalan lo que se conoce como *limitaciones alfa* y las barras dobles denotan *limitaciones beta*. Éstas cercenan las bifurcaciones que no pueden incidir en el resultado final.

Una limitación *alfa* se produce en el nudo E, que no es necesario evaluar nunca, como tampoco a ninguno de sus descendientes, si hubiera alguno. Para cuando llegamos al nudo E sabemos que el nudo C obtiene un marcador de 15, pero en el nudo D el oponente puede forzarnos a bajar a 10. No tiene ningún sentido averiguar si nos podemos ver

El estado del juego



El diagrama ilustra el estado actual del juego en una partida de tres en raya tras seis movimientos (seguidamente han de jugar las cruces). Se puede construir un sencillo árbol de juego para rastrear los tres turnos finales del juego considerando la cantidad de opciones existentes en cada etapa. El primer nivel tiene tres nudos, que corresponden a los tres movimientos posibles (a los cuadrados 1, 2 y 7) que pueden efectuar las cruces. Cuando les toca jugar a los círculos, sólo les quedan dos cuadrados entre los que escoger, con tres posibilidades (de las cuales habrá disponibles dos después de que las cruces efectúen un movimiento). El siguiente nivel tiene, por lo tanto, seis nudos. El nivel inferior indica situaciones en las que se han utilizado todos los cuadrados y se compone de nudos "terminales"

Selección de cuadrados
Habiendo construido el árbol del juego, a cada nudo terminal se le puede asignar un valor: 1 para ganan las cruces, 0 para un empate y −1 para ganan los círculos. Podemos entonces ir retrocediendo a través del árbol asignando valores a los otros nudos. Si tomamos el nudo más a la derecha del primer nivel, se llega al valor −1 considerando los dos nudos debajo de éste, que poseen valores de 0 y −1. Dado que el valor lo determinan los círculos (les toca jugar a los círculos), se seleccionará el valor mínimo, es decir, −1. Retrocediendo hasta arriba del árbol al movimiento actual, las cruces deben elegir el valor máximo de los tres disponibles. En este ejemplo, las cruces deberían dar el juego por empatado seleccionando el cuadrado 2 para el siguiente movimiento (las otras dos opciones darían como ganador a los círculos)

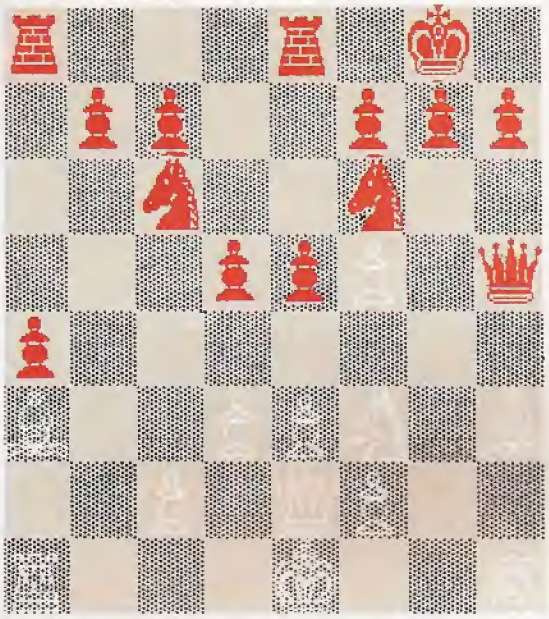
Liz Dixon

Movimientos de la partida

1	d2-d3	e7-e5
2	Cb1-d2	Cb8-c6
3	g2-g3	Cg8-f6
4	Af1-g2	Af8-c5
5	e2-e3	d7-d5
6	Cg1-e2	0-0
7	a2-a3	Ac8-f5
8	b2-b3	Cf6-g4?!
9	h2-h3	ng4-f6
10	Ac1-b2	Dd8-d6
11	g3-g4	Af5-e6
12	Ce2-g3	a7-a5
13	Dd1-e2	Cf6-d7
14	Cg3-f5	Ae6xf5
15	g4xf5	Cd7-f6
16	h3-h4	Tf8-e8
17	Ag2-h3	a5-a4
18	b3-b4	Ac5xb4!?
19	a3xb4	Dd6xb4
20	Ab2-a3	Db4xh4
21	Cd2-f3	Dh4-h5
22	Cf3-d2	Dh5xe2+?
23	Re1xe2	b7-b5
24	c2-c3	h7-h6
25	Ah3-g2	Ta8-a5
26	Ta1-b1	Te8-b8
27	Tb1-b2	Tb8-b6
28	Th1-b1	Rg8-h7
29	Aa3-c5	Tb6-b8
30	Ac5-a3	Cc6-a7
31	Cd2-f3	Cf6-d7
32	Cf3-e1	c7-c6
33	Cd1-c2	Tb8-a8
34	Cc2-b4	Ta8-d8
35	Cb4-a2	Ca7-c8
36	c3-c4!	d5xc4
37	d3xc4	b5xc4
38	Ag2xc6	Cc8-a7
39	Ac6-e4	Cd7-f6
40	Aa3-e7	Td8-c8
41	Ae7xf6	g7xf6
42	Tb1-b6!	c4-c3
43	Tb6xf6	Rh7-g7
44	Tf6-b6	a4-a3
45	Tb1-g1+	

En este punto el programa indicó que estaría 2.4 peones abajo y los programadores abandonaron

Posición de las piezas tras el movimiento 21



Esta partida forma parte de una serie en la que el programa de ajedrez por ordenador más potente del mundo, el *Cray Blitz*, fue derrotado 4-0 por David Levy, de Intelligent Software, tras una apuesta de 5 000 dólares entre el señor Levy y los programadores del *Blitz*. Aunque el programa ha obtenido la categoría ajedrecista de National Master, el juego demuestra que aún es necesario muchísimo trabajo para que los ordenadores supongan un serio desafío a los mejores jugadores humanos del mundo

obligados a bajar aún más, puesto que obviamente esta ruta es menos deseable que la ruta a través del nudo C. Por tanto, se pueden sustraer de toda consideración los otros descendientes del nudo F.

En el nudo I se aplica el mismo razonamiento pero al contrario. Para cuando llegamos allí, sabemos que G produce un marcador de 20. El nudo H, con 25, parece mejor, pero MINI (y no MAX) elige entre los nudos G y J, y claramente se inclinará por G. Por consiguiente, no hay necesidad de ver si I es aún más prometedor, puesto que a MAX jamás se le permitirá llegar allí.

Podemos expresar estas ideas desde el punto de vista de un árbol genealógico. MAX es un machista que piensa que el nudo C, por ejemplo, es el tío de los nudos D y E, que son ambos hijos del mismo padre. MINI, por su parte, es una feminista y, en cuanto a ella respecta, G es la tía de las hermanas H e I, cuya madre es J. En la medida en que usted no se sienta confundido por nudos que cambian de sexo en niveles alternativos, esta analogía nos permitirá explicar con precisión la regla *alfabeta*:

- Apenas descubre MAX un hijo que es *peor* que cualquiera de sus tíos, ignora a los otros hermanos de ese hijo.
- Apenas descubre MINI una hija que es *mejor* que cualquiera de sus tías, ignora a las otras hermanas de esa hija.

En el mejor de los casos, el algoritmo *alfa-beta* sólo examina dos veces más la raíz cuadrada de la cantidad de nudos terminales del árbol del juego, en comparación con la simple minimaximización. En el peor de los casos, examina la misma cantidad, y de forma ligeramente más lenta. Para evitar el primero de nuestros dos casos, es importante generar los hermanos y hermanas de cada nivel por un orden sensato. En los niveles maximizadores, se los debe generar por orden de primero el mejor, y en los niveles minimizadores, por orden de primero el peor (primero el mejor para el adversario).

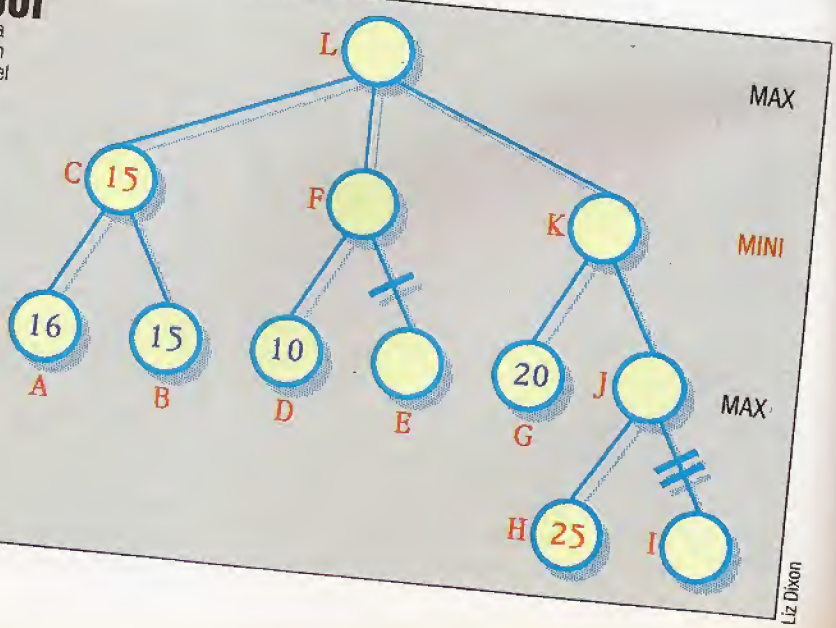
Para ilustrar los importantes conceptos de la búsqueda arborescente, presentamos un juego artificial que comprende técnicas de búsqueda casi puras. Ello significa que los detalles de representación del tablero, generación de movimientos y evaluación estática (que son cruciales para el éxito de cualquier programa verdadero pero que también son específicos de cada juego) no oscurecen la simplicidad esencial del procedimiento *alfa-beta*.

No hay ni tablero ni ninguna pieza: el estado del juego se describe completamente mediante un único número, retenido como V%. Para el ordenador, el objetivo del juego es que V% alcance el valor 255, y para usted conseguir que sea menor que −255.

A cada turno el jugador puede elegir entre aplicar una de cuatro funciones (A,B,C,D), listadas entre las líneas 1030 y 1060. Es posible alterarlas para crear diferentes versiones del juego, como,

Poda del árbol

El podado *alfa-beta* perfecciona el método de minimaximización básico suprimiendo del árbol del juego las ramificaciones redundantes. Los cortes *alfa* (señalados aquí mediante una barra simple) se efectúan cuando en el movimiento de MINI ya se ha hallado una ruta mínima y ya no son necesarias otras búsquedas. Por el contrario, los cortes *beta* (señalados mediante una barra doble) se realizan tras hallarse una ruta máxima en el movimiento de MAX



Liz Dixon

por ejemplo, hacer que le resulte más difícil al ordenador.

El juego es trivial pero ilustra la estrategia de búsqueda sin que detalles ajenos dificulten su apreciación. Además, es sumamente matemático, de modo que el ordenador posee una ventaja incorporada. En esta versión, la minimaximización *alfa-beta* se basa en gran medida en el uso de funciones recursivas con parámetros y variables locales. Los parámetros son los siguientes:

VV% Estado actual del juego
A% Mejor *Alfa* hasta ahora a este nivel
B% Peor *Beta* hasta ahora a este nivel
D% Indicador de profundidad

Dado que el programa implica recursión, aquí sólo incluimos una versión en BASIC BBC.

El juego de los números

```
10 REM ********************************************
11 REM **                Listado 3.1:              **
12 REM **          EL JUEGO DE LOS NUMEROS         **
13 REM ********************************************
50 MODE 7
100 REM — Juego para demostrar busqueda:
120 GOSUB 1000   :REM inicializacion
130 GOSUB 1600   :REM instrucciones
150 REPEAT
160   GOSUB 2000   :REM preparar nuevo juego
170   INPUT "Quien va primero (1=Tu, 2=Yo)  ",H1%
180   IF H1%<1 OR H1%>2 THEN GOTO 170
200   REM — bucle principal:
210   REPEAT
220     IF H1%=1 THEN GOSUB 3000
230     REM — turno de la persona.
240     GOSUB 3500   :REM visualizacion tablero
250     H1%=1 : REM siempre 1 tras 1.er ciclo
260     GOSUB 4000 : REM verificar ganador
270     IF EG%=0 THEN GOSUB 5000
280     REM — turno del ordenador.
290     GOSUB 3500 : REM mostrar estado del juego
300     GOSUB 4000 : REM verificar final del juego
310     UNTIL EG%<>0 OR M%>33
320   REM — ultimo movimiento:
330   GOSUB 6000 : REM felicitaciones
340   PRINT"Otra partida (N=No)  ";
350   Y$=GET$
360   UNTIL Y$="N" OR Y$="n"
365 PRINT
370 PRINT"Hasta otra, y gracias por jugar!"
400 END
444 :
500 DEF FNmaximov(VV%,A%,B%,D%)
505 REM --------- mi turno:
510 LOCAL P%,E%, KEEP%
515 IF D%>=MD% OR ABS(VV%)>HI% THEN=VV% : REM valor estatico
520 REM si no profundizar mas:
525 P%=0
530 REPEAT P%=P%+1
535   H%=P%: V%=VV%: GOSUB 5500:REM efectuar movimiento
536   IF D%=1 THEN PRINT CHR$(H%+64);"=";
540   E%=FNminimov(V%,A%,B%,D%+1)
545   IF E%>A% THEN A%=E%: KEEP%=P%
546   IF D%=1 THEN PRINT E%;" ";
550   UNTIL P%>3 OR A%>=B%
555 IF A%>BV% AND D%=1 THEN BV%=A%: HH%=KEEP%
556 REM guardar el mejor hasta ahora.
560 =A%
565 REM volver con max A%
570 :
700 DEF FNminimov(VV%,A%,B%,D%)
710 REM --------- turno del otro:
720 LOCAL E%,P%
730 IF D%>=MD%OR ABS(VV%)>HI% THEN=VV%
740 P%=0
750 REPEAT P%=P%+1: H%=P%
755   V%=VV%: GOSUB 5500: REM efectuar movimiento
760   E%=FNmaximov(V%,A%,B%,D%+1)
770   IF E%<B% THEN B%=E%
780   UNTIL P%>3 OR B%<=A%
790 =B%
796 REM vuelve con el valor de B% mas bajo
999 :
1000 REM -- rutina inicializadora:
1001 BL$=""
1002 @ %=4
1020 REM -- las 4 funciones:
1030 DEF FNA(X%)=2*X%-7
1040 DEF FNB(X%)=X% DIV 2+1
1050 DEF FNC(X%)=-4*X%+17
1060 DEF FND(X%)=3*X%-4
1070 LO%=-255: HI%=255
1150 RETURN
1160
1600 REM -- rutina instrucciones:
1610 CLS:PRINT
1620 PRINT""Bienvenido al Juego de los Numeros!"
1630 PRINT"Si no conoces las reglas,"
1635 PRINT"LEETE EL CAPITULO!"'
1636 PRINT"Nota: Yo maximizo : tu minimizas."
1640 PRINT"Para ver el efecto de un movimiento, digita:"
1645 PRINT"A, B, C, o D. Para efectuarlo, digita X."
1650 PRINT:PRINT"Buena suerte!";CHR$(7)
1660 RETURN
1670
2000 REM -- Rutina de preparacion:
2010 M%=0 : REM movimientos
2020 V%=RND(15)-8: REM estado inicial.
2050 EG%=0
2060 PRINT"Estado inicial=";V%
2100 RETURN
2110
3000 REM -- Movimiento de la persona
3010 M%=M%+1
3020 PRINT
3030 REPEAT
3040   PRINT"Que movimiento haces ? ";
3050   H$=GET$: PRINT H$;
3060   IF H$="A" THEN PRINT FNA(V%): H%=1
3070   IF H$="B" THEN PRINT FNB(V%): H$=2
3080   IF H$="C" THEN PRINT FNC(V%): H$=3
3090   IF H$="D" THEN PRINT FND(V%): H$=4
3100   UNTIL H$="X"
3120 GOSUB 5500 : REM elegir H%
3150 RETURN
3160 :
3500 REM -- rutina visualizacion del tablero:
3520 CLS: PRINT
3522 PRINT"      Movimiento ";M%;" --> ";
3523 IF M%<1 THEN RETURN
3525 PRINT CHR$(64+H%);
3530 PRINT "  =  ";V%
3535 RETURN
3700
4000 REM -- Rutina comprobacion ganador (sobre M$):
4001 IF M%<1 THEN RETURN
4010 EG%=0
4020 IF V%<LO% THEN EG%=-1
4030 IF V%>HI% THEN EG%=1
4040 RETURN
4050 :
5000 REM -- Rutina movimiento del ordenador:
5005 W%=V% : REM guardar estado actual.
5010 M%=M%+1
5015 MD%=6 : REM max profundidad
5020 IF M%<4 THEN MD%=4
5030 IF M%>8 THEN MD%=8
5040 GOSUB 5200 : REM -->H%
5045 V%=W% : REM restablecer estado.
5050 GOSUB 5500 : REM hacerlo.
5070 RETURN
5080 :
5200 REM -- Seleccion de movimiento:
5210 BV%=LO% : D%=0
5220 BV%=FNmaximov(V%,LO%,HI%,1)
5230 H%=HH%
5240 PRINT "Pulsar cualquier tecla para continuar: ";
5244 C%=GET
5250 RETURN
5270 :
5500 REM -- Rutina efectuar movimiento (H% : V%):
5505 ON H% GOTO 5510,5520,5530,5540
5510 V%=FNA(V%): RETURN
5520 V%=FNB(V%): RETURN
5530 V%=FNC(V%): RETURN
5540 V%=FND(V%): RETURN
5550 :
6000 REM -- Rutina de felicitaciones:
6010 PRINT "EL JUEGO HA TERMINADO!"
6020 IF EG%>0 THEN PRINT "He ganado yo!!"
6030 IF EG%<0 THEN PRINT "Bien hecho!"
6040 IF EG%=0 THEN PRINT "Ha sido empate"
6050 RETURN
6600 :
```

Prólogo

Parece que haya pasado mucho tiempo desde que los japoneses afirmaron que cambiarían el curso de la tecnología del ordenador invirtiendo dinero y trabajo de investigación en su proyecto de ordenadores de la quinta generación. Quizás una de las decisiones más significativas que adoptaron fue utilizar el PROLOG, lenguaje de programación poco conocido, como el "lenguaje central" de las máquinas de bases de datos inteligentes y de elevado rendimiento que prevén.

PROLOG representa las siglas de "*pro*gramming in *log*ic" (programar en lógica) y constituye una concreción buena pero imperfecta de ese ideal. Pero ¿por qué habría de ser deseable programar en lógica? Existen muchas lógicas que se pueden aplicar para describir el mundo y aspectos del mismo. Con algunas de éstas usted ya estará familiarizado, como la matemática, mientras que otras pueden parecerle bastante ajenas, como ciertas doctrinas filosóficas. El cálculo de predicado de primer orden es una lógica bastante parecida a la que utilizamos para el pensamiento y el análisis cotidiano, aunque posee su propia notación y ciertas restricciones. Se puede concebir el predicado como una relación entre cosas. En la frase "a Juan le gusta Ana", el predicado es "gusta" Podríamos escribir esto como "gusta (Juan, Ana)", que es menos inteligible pero más claro en cuanto a qué es el predicado y cuáles son sus argumentos. Para expresar que Juan es varón, podríamos escribir "varón(Juan)", donde "varón" es un predicado que toma un argumento ("Juan"). De modo similar, "mujer(Ana)", significa que Ana es una mujer.

Lo que tenemos aquí son simplemente declaraciones de hechos pero podemos ampliar la lógica para mostrar cómo algunos hechos *implican* otros. Utilizando la lógica de predicado, podemos describir el mundo en términos de hechos e implicaciones y usar nuestra descripción para deducir hechos nuevos a partir de los antiguos. Ello lo hacemos mediante la introducción de variables. Las variables lógicas son muy similares a aquellas que usted ya conoce a través del BASIC o algunos otros lenguajes de programación, con la excepción de que su esfera de acción se limita a la cláusula (hecho o implicación) en la que aparecen, en vez de a todo el conjunto de cláusulas. Esto significa que el Juan a quien le gusta Ana tal vez sea un Juan diferente del Juan que es un varón.

Ahora podríamos escribir un hecho tal como "mujer(X) →gusta(Juan, X)". La flecha significa "implica que", de modo que podemos leer esto como una regla que afirma que "el hecho de que X sea mujer implica que a Juan le gusta X". En castellano corriente, la regla se lee como "a Juan le gusta X si X es mujer". Esta regla es un ejemplo de un tipo especial de cláusula del cálculo de predicado que se denomina *cláusula horn* (trompeta). Las cláusulas *horn* poseen una sentencia como encabezamiento (el *consecuente*), que es verdadera sólo si todas las sentencias del cuerpo (los *antecedentes*) lo son.

A si B y C y D

es una cláusula *horn* con la cabeza A y el cuerpo B, C y D. Un hecho simple se puede considerar como un consecuente sin antecedentes y se lo presume verdadero.

Con lógica podemos expresar el programa que deseamos escribir como un conjunto de hechos y reglas que describen las cosas en las que estamos interesados. Esta descripción es una forma bastante similar a aquella en que concebimos realmente el problema. Para "ejecutar" nuestro programa lógico intentamos demostrar la veracidad o falsedad de algún enunciado. Si éste es un hecho simple, entonces podemos suponer que es verdadero sin ningún otro esfuerzo. Si es el consecuente de alguna regla, entonces debemos abocarnos a demostrar la veracidad de todos sus antecedentes antes de que podamos afirmar que es verdadero. Por tanto, si deseamos saber si Juan es varón, intentamos demostrar la sentencia "varón(Juan)", que sabemos que es verdadera porque es un hecho que ya poseemos. Si queremos saber si a Ana le gusta Juan, primero necesitamos probar las sentencias "varón(Juan)" y "gusta(Juan, Ana)".

El PROLOG lo desarrolló A. Colmerauer en la Universidad de Marsella a principios de los años setenta, basándose parcialmente en el trabajo de Bob Kowalski, que en la actualidad se desempeña en el Imperial College de Londres. Utiliza sólo las cláusulas *horn* de la lógica de predicado y una notación similar a la que hemos

Liz Heaney

Preludio de "Complementos al PROLOG"

Al objeto de implementarse con eficacia, el PROLOG tiende a exigir generosas cantidades de RAM, y son raras las versiones del lenguaje que puedan ejecutarse en un ordenador personal. Los usuarios del Spectrum, no obstante, pueden considerar la adquisición del MICRO-PROLOG, escrito por Logic Programming Associates y distribuido por Sinclair en cassette. Los usuarios de otros micros pueden mantener vivas sus esperanzas, a la vista de que el PROLOG está captando rápidamente la atención de los productores de software y ya existen planes para versiones del lenguaje destinadas al BBC Micro y al Enterprise. El MICRO-PROLOG Spectrum difiere en muchos sentidos del DEC-10 PROLOG estándar, la versión utilizada en los ejemplos a lo largo de nuestra serie. No obstante, iremos imprimiendo una serie de breves *Complementos al* PROLOG, para que los usuarios del Spectrum puedan entrar listados directamente con MICRO-PROLOG. Las diferencias más importantes las explicaremos más adelante

mostrado más arriba. Se podría haber implementado el cálculo de predicado en su totalidad, pero el PROLOG, como todos los lenguajes de programación, representa un equilibrio entre eficiencia de proceso y poder de expresión. Se está desarrollando un gran trabajo de investigación para refinar este equilibrio, pero en su forma actual el PROLOG parece bastante estable.

Lo más cercano a un PROLOG estandarizado es una versión denominada DEC-10 PROLOG (porque se implementó por primera vez en un ordenador central de Digital Equipment Corporation), y la biblia de los usuarios del PROLOG es un libro de Clocksin y Mellish que con toda sencillez se titula *Programming in* PROLOG y describe esta versión y ofrece, paralelamente, muchos detalles prácticos. Casi todos los PROLOG actuales, incluyendo el C-PROLOG, están modelados según este estándar, aunque abundan los complementos y los dialectos. Para el micro, hay dos versiones que marchan a la vanguardia: la de Expert Systems, que se acerca mucho al estándar, y el MICRO-PROLOG, de Logic Programming Associates, que difiere en cuanto a sintaxis y estructura interna.

Las implementaciones de PROLOG utilizan muchísima memoria y por este motivo no caben con comodidad en micros con menos de 64 K de RAM. Sin embargo, existe el MICRO-PROLOG para el Spectrum y continúan apareciendo nuevas implementaciones.

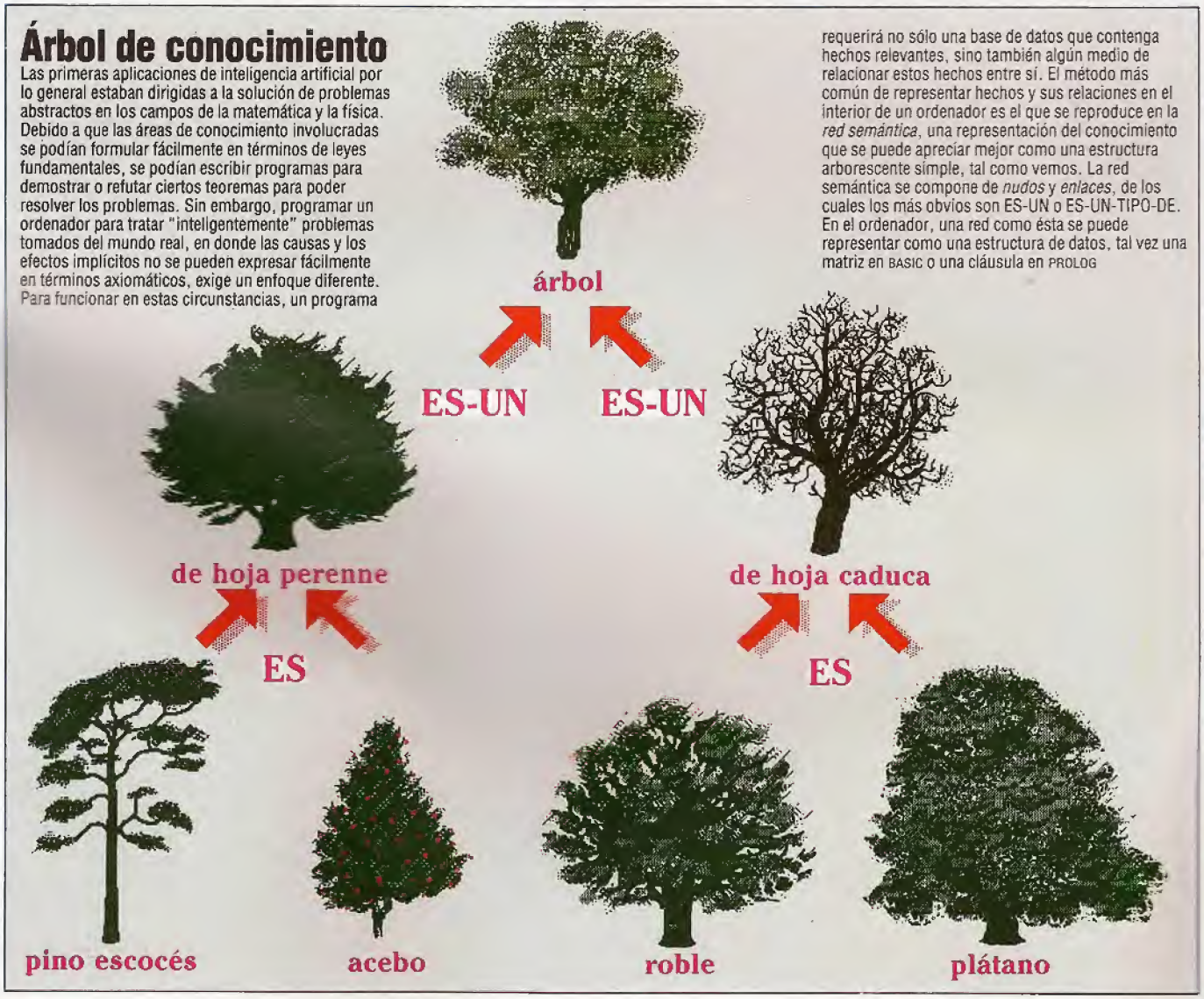
Los programas en PROLOG no exhiben el familiar flujo de control en el cual se ejecuta la primera sentencia del programa, luego la segunda, y así sucesivamente hasta la última, con ocasionales bifurcaciones y bucles a lo largo del camino. En cambio, el PROLOG utiliza una técnica "*de retroceso*".

Para resolver un interrogante, el PROLOG va trabajando hacia abajo a través de una cadena de reglas, planteándose a sí mismo un nuevo objetivo a demostrar en cada ocasión. Si un camino determinado a través de la cadena demuestra ser improductivo, el PROLOG "retrocederá" hasta un punto de elección anterior y después se bifurcará en una nueva dirección.

Esta forma de proceder le confiere al PROLOG un "tacto" muy diferente en comparación con cualquier otro lenguaje de programación. Los defensores de la programación lógica resaltan la naturaleza declarativa de un programa en PROLOG. Es decir, la lectura de una regla como si fuera una cláusula del cálculo de predicado de primer orden; por ejemplo, X es tío de Y si X es varón y X es el hermano de Z y Z es el padre de Y. Y el PROLOG siempre se puede leer en el estilo procedural más familiar (para demostrar que X es un tío de Y, demostrar primero que X es varón, después demostrar que X es el hermano de Z, y después demostrar que Z es el padre de Y). La disponibilidad de la lectura declarativa es algo de lo que carecen la mayoría de los otros lenguajes y, ciertamente, es una característica muy valiosa para ayudarlo a usted a comprender y, por lo tanto, a diseñar y comprobar sus programas.

Árbol de conocimiento

Las primeras aplicaciones de inteligencia artificial por lo general estaban dirigidas a la solución de problemas abstractos en los campos de la matemática y la física. Debido a que las áreas de conocimiento involucradas se podían formular fácilmente en términos de leyes fundamentales, se podían escribir programas para demostrar o refutar ciertos teoremas para poder resolver los problemas. Sin embargo, programar un ordenador para tratar "inteligentemente" problemas tomados del mundo real, en donde las causas y los efectos implícitos no se pueden expresar fácilmente en términos axiomáticos, exige un enfoque diferente. Para funcionar en estas circunstancias, un programa requerirá no sólo una base de datos que contenga hechos relevantes, sino también algún medio de relacionar estos hechos entre sí. El método más común de representar hechos y sus relaciones en el interior de un ordenador es el que se reproduce en la *red semántica*, una representación del conocimiento que se puede apreciar mejor como una estructura arborescente simple, tal como vemos. La red semántica se compone de *nudos* y *enlaces*, de los cuales los más obvios son ES-UN o ES-UN-TIPO-DE. En el ordenador, una red como ésta se puede representar como una estructura de datos, tal vez una matriz en BASIC o una cláusula en PROLOG



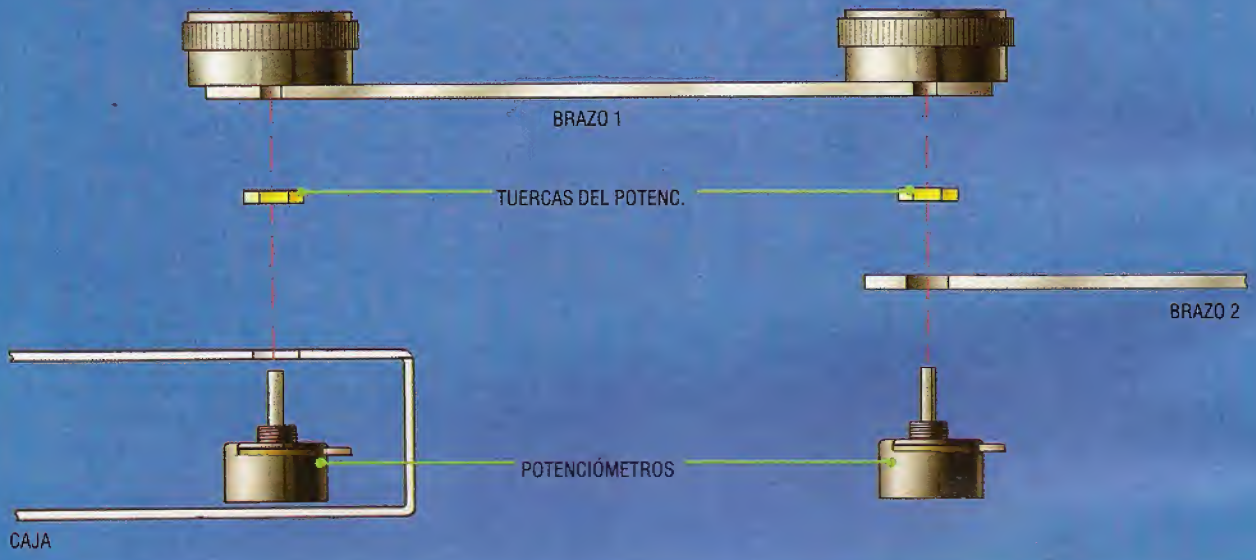
Ian McKinnell

Ajustando la mira

Ahora montaremos los componentes del brazo y agregaremos los potenciómetros y la mira del trazador

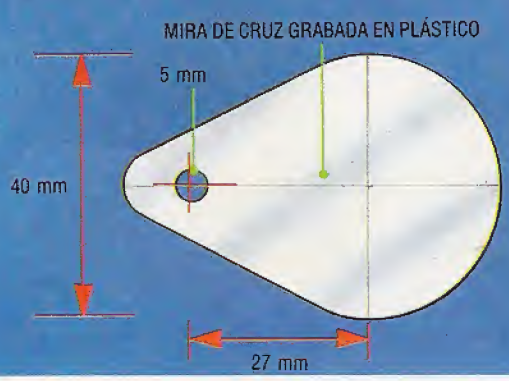
Paso 1: Ensamblar el brazo

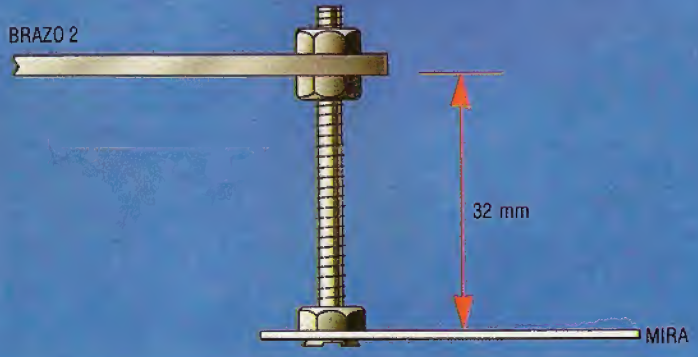
Las dos piezas del brazo que cortamos en el último capítulo se engoznan mediante un par de potenciómetros, uno de los cuales se monta en la tapa de la caja del componente plástico y el otro a través de los extremos del brazo conector. Corte el husillo de cada potenciómetro de modo que sólo sobresalga 15 mm del cuerpo. Fije un potenciómetro en la tapa de la caja y otro en el brazo 2, ajustándolos en su sitio con las tuercas. Observe que los puntos de conexión del potenciómetro de la caja deben apuntar hacia el lado de la caja más cercano, mientras que los del otro deben apuntar a lo largo del brazo. Empuje los husillos en las perillas correspondientes, ya montadas en el brazo 1, y fíjelos en su sitio ajustando los tornillos empotrados. Cada husillo debe posicionarse de modo que el borde plano quede afianzado mediante el tornillo empotrado

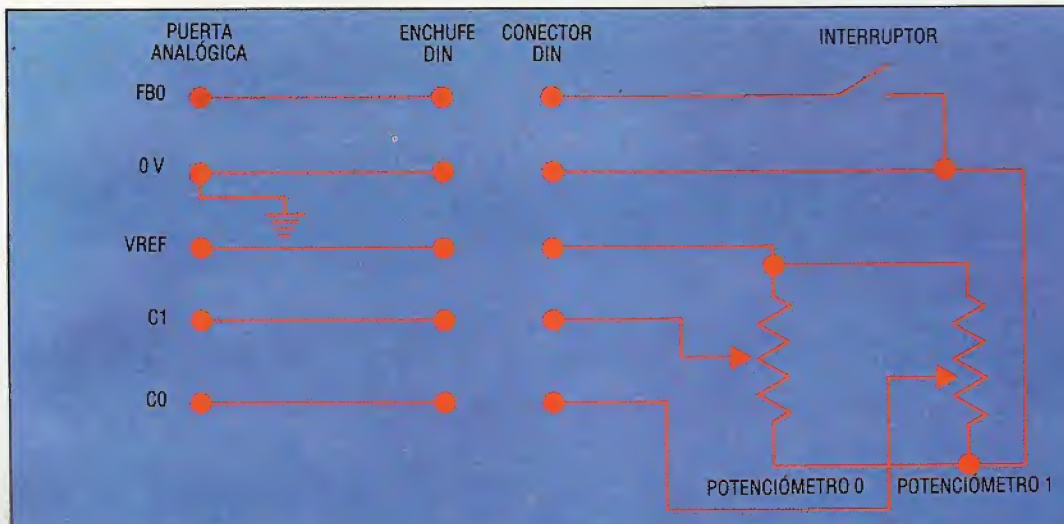


Paso 2: Montar la mira

Corte una mira de un trozo de plástico transparente relativamente delgado, como el que se utiliza para los estuches de cassettes de audio. La forma exacta de la mira no es importante, pero los ejes deben trazarse sobre la superficie de modo que se crucen (ver ilustración) y queden a 27 mm del centro del agujero de montaje. Fije la mira en el extremo libre del brazo 2 mediante el perno M5 y tres tuercas. Establezca la altura inicial en 32 mm. Esta disposición permitirá cambiar fácilmente la altura

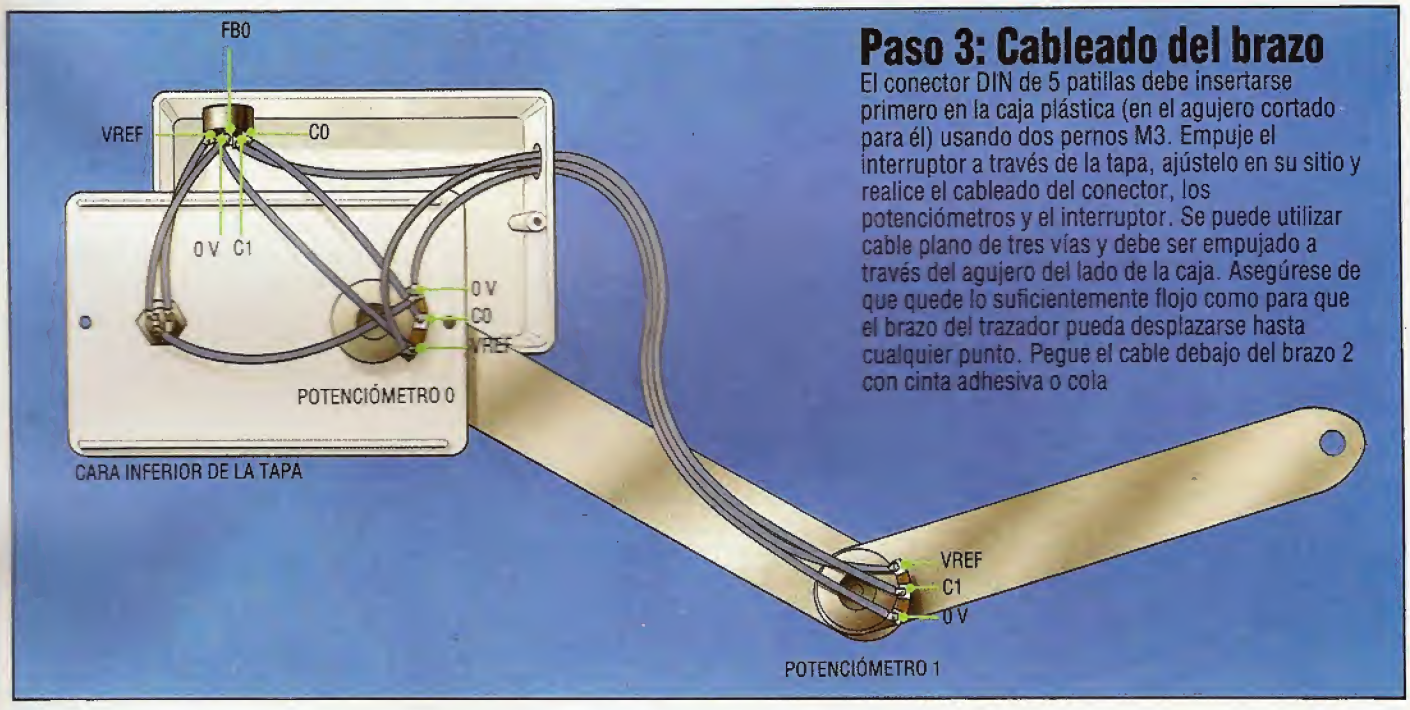






El circuito

He aquí el circuito completo para el trazador digital. Observe que el interruptor a presión está conectado a FB0 en la puerta analógica. Esta corresponde a una entrada de botón de disparo que se puede detectar mediante software. El voltaje de referencia utilizado en la conversión de analógico a digital se suministra a través de las dos pistas de los potenciómetros; las conexiones del potenciómetro central se vuelven a enviar a los canales ADC 0 y 1 para la conversión a forma digital. Utilizando esta información, podemos calcular los ángulos del brazo y la posición de la mira del trazador

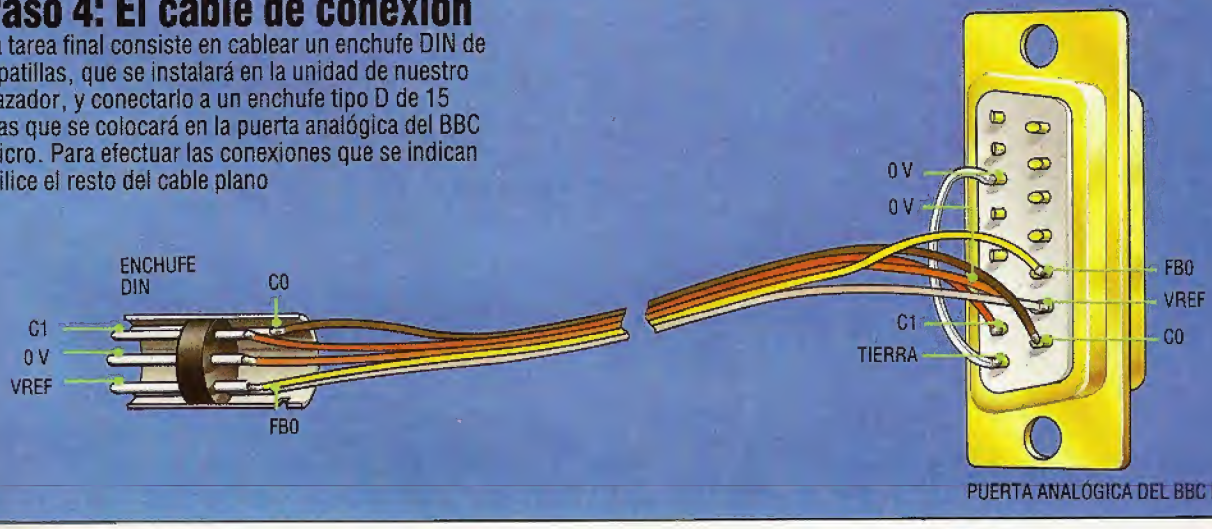


Paso 3: Cableado del brazo

El conector DIN de 5 patillas debe insertarse primero en la caja plástica (en el agujero cortado para él) usando dos pernos M3. Empuje el interruptor a través de la tapa, ajústelo en su sitio y realice el cableado del conector, los potenciómetros y el interruptor. Se puede utilizar cable plano de tres vías y debe ser empujado a través del agujero del lado de la caja. Asegúrese de que quede lo suficientemente flojo como para que el brazo del trazador pueda desplazarse hasta cualquier punto. Pegue el cable debajo del brazo 2 con cinta adhesiva o cola

Paso 4: El cable de conexión

La tarea final consiste en cablear un enchufe DIN de 5 patillas, que se instalará en la unidad de nuestro trazador, y conectarlo a un enchufe tipo D de 15 vías que se colocará en la puerta analógica del BBC Micro. Para efectuar las conexiones que se indican utilice el resto del cable plano



Kevin Jones

Revolución gráfica

Proseguimos nuestro estudio de los gráficos en 3 dimensiones del Commodore 64, analizando esta vez la conversión del listado en BASIC

El programa híbrido (*Prueba I-Rot* y *Hexa II-Rot*) anteriormente proporcionado se ejecuta con relativa rapidez. Pero es claro que el tener que inspeccionar la matriz de adyacencia E%(I,J) (que define los nodos que han de conectarse en la figura de líneas) con el solo objeto de determinar cuáles son los puntos que deben trazarse, ralentiza la marcha del programa.

Para darle mayor celeridad necesitamos codificar el resto del bucle clave en BASIC del programa *Cubo rotatorio* en código máquina. El resultado vale la pena por la velocidad ganada.

Para tratar los siguientes cálculos matemáticos:

```
X1%=X(I)+159:Y1%=199-(Z(I)+100)
X2%=X(J)+159:Y2%=199-(Z(J)+100)
```

que encontramos en las líneas 1640 y 1650 del programa original en BASIC, necesitamos otras llamadas al intérprete.

Esencialmente ambas líneas del BASIC toman una variable de coma flotante (p. ej., X(I)) y le añaden 159 (en coma flotante también) antes de aceptar la parte entera y almacenarla como X1% en dos bytes.

Las llamadas al intérprete necesarias para lograr esto son:

- FLPINT (dirección de la llamada $B1AA):

Esta rutina toma la parte entera del número en FAC y da el resultado (dentro del intervalo −32767 a 32767) en forma de byte *lo/hi* en los registros Y y A. Nótese aquí el orden *lo/hi* inusual, al revés que en muchas rutinas de intérprete.

- SNGFT (dirección de la llamada $B3A2):

Esta rutina toma un entero de un solo byte (un número entre 0 y 255) del registro Y, y lo coloca en FAC en coma flotante.

SNGFT se emplea en la subrutina ESTABLECIMIENTO del listado en assembly (línea 5150). Por ejemplo, el valor decimal 159 es colocado en el registro índice Y y se llama a SNGFT para convertirlo y situarlo en FAC.

Después de esto, MOVMF se encarga de poner el resultado en los cinco bytes de MEM1. Cuando deseamos sumar 159 sabemos que está disponible en MEM1.

Los problemas más importantes que surgen en la conversión a código máquina del bucle del BASIC se refieren a la especificación de determinados elementos en las tablas que definen la figura rotatoria. Puede que en algunos casos el cálculo de los punteros de la tabla resulte difícil. Las tablas de ordenadas X(I),Y(I),Z(I) no presentan dificultades especiales, ya que en cada tabla sencillamente añadimos cinco bytes al puntero para obtener la dirección del siguiente elemento.

La tabla E%(I,J), al ser bidimensional, es otra cosa.

Se ordena en la memoria secuencialmente de la siguiente manera:

```
E%(0,0)E%(1,0)E%(2,0)...E%(NP,0)
E%(0,1)E%(1,1)E%(2,1)...E%(NP,1), etc.
```

Es decir, la tabla se compone de bloques de memoria de 2×(NP+1) bytes de longitud cada uno, correspondiente a los valores del segundo subíndice, tomando cada elemento dos bytes (la tabla es de números enteros).

Nuestra intención es reflejar el código BASIC con la mayor exactitud en código máquina de tal modo que los bucles I,J (rastreadores de E%(I,J)) serán los siguientes:

```
FOR I=1 TO NP
FOR J=1 TO I
```

Esto significa que el cambio en el puntero para lograr un equivalente en código máquina de NEXT I, que permita incluir elementos con un primer subíndice cero, resulta algo complejo. Para hacer que la figura gire, se ha de acceder a los elementos de E%(I,J) en el siguiente orden:

```
E%(1,1)
E%(1,2)E%(2,2)
E%(1,3)E%(2,3)E%(3,3)
E%(1,4)E%(2,4)E%(3,4)E%(4,4), etc.
```

Un cálculo rápido muestra que hay que añadir 2×(NP+1) al puntero cada vez que se incrementa I. La mejor manera de hacer esto en el código máquina del 6502 es emplear el direccionamiento indirecto para acceder a E%(I,J).

El código sería:

```
LDY JINDEX
LDA (ZTEMP),Y
```

siendo ZTEMP la página cero, un puntero de dos bytes, y JINDEX sirve para seguir la pista de J. También ZTEMP debe ser incrementado a cada incremento de J. Los incrementos de ZTEMP y del registro Y se acompañan de un aumento del desplazamiento en dos bytes a cada incremento de J. El resultado final es que ZTEMP debe quedar incrementado en (2×NP+1)−(I−1) a cada iteración del bucle I.

La longitud del bloque es decrementada en (I−1) porque ZTEMP ha sido ya incrementada (I−1) veces en el bucle J ejecutado.

La expresión del cálculo del desplazamiento quiere decir que ZTEMP apunta al byte correcto después de ser incrementado I.

Por último, sería útil poder llamar a una rutina intérprete para que nos coloque la variable E%(I,J).

Tal rutina existe, pero resulta por desgracia tan sibilina (pues trata todo tipo de variables posibles) y tan lenta, que es preferible que calculemos nosotros mismos el desplazamiento de la dirección de E%(1,1).

Rutinas rotatorias

Los listados que siguen dan el código fuente del programa en código máquina que hace girar una figura de líneas, definida por la tabla E%(I,J) en una pantalla de alta resolución. El programa emplea las rutinas intérprete y métodos de posicionamiento de las variables del BASIC examinadas en el texto. Conseguimos ahora nuestro objetivo inicial de convertir el bucle original en BASIC que inspecciona las tablas para dar la figura rotatoria. Se incluyen además un cargador en BASIC para este código fuente, junto con un programa de prueba en BASIC que establece las variables en código máquina y llama a la rutina. El programa de prueba puede ser empleado con una versión ensamblada del código fuente, HEXA II-ROT.O cargando y ejecutando el programa cargador, si pulsamos NEW y finalmente cargamos y ejecutamos el programa de prueba. En este segundo caso la línea 1030 será omitida.

Cargador en BASIC

```
1000 REM *** INSERTAR HEXA II-ROT EN C/M ***
1010 REM ********************************
1020 DATA72,138,72,152,72,32,101,199
1030 DATA173,83,197,172,84,197,32,162
1040 DATA187,173,75,197,172,76,197,32
1050 DATA40,186,162,94,160,197,32,212
1060 DATA187,173,87,197,172,88,197,32
1070 DATA162,187,173,77,197,172,78,97
1080 DATA32,40,186,169,94,160,197,32,80
1090 DATA184,162,94,160,197,32,212,187
1100 DATA173,87,197,172,88,197,32,162
1110 DATA187,173,75,197,172,76,197,32
1120 DATA40,186,162,99,160,197,32,212
1130 DTA187,173,83,197,172,84,197,32
1140 DATA162,187,173,77,197,172,78,197
1150 DATA32,40,186,169,99,160,197,32
1160 DATA103,184,162,99,160,197,32,212
1170 DATA187,169,94,160,197,32,162,187
1180 DATA174,83,197,172,84,197,32,212
1190 DATA187,169,99,160,197,32,162,187
1200 DATA174,87,197,172,88,197,32,212
1210 DATA187,206,93,197,240,31,169,5,24
1220 DATA109,83,197,141,83,197,144,3
1230 DATA238,84,197,169,5,24,109,87,197
1240 DATA141,87,197,144,3,238,88,197,76
1250 DATA112,197,169,1,141,0,193,141,1
1260 DATA193,141,2,193,32,14,193,32,101
1270 DATA199,169,1,141,81,197,141,82
1280 DATA197,173,79,197,133,253,173,80
1290 DATA197,133,254,172,82,197,177,253
1300 DATA208,3,76,207,198,173,83,197
1310 DATA172,84,197,32,162,187,169,94
1320 DATA160,197,32,103,184,32,170,177
1330 DATA140,0,195,141,1,195,173,85,197
1340 DATA172,86,197,32,162,187,169,94
1350 DATA160,197,32,103,184,32,170,177
1360 DATA140,2,195,141,3,195,173,89,197
1370 DATA172,90,197,32,162,187,169,99
1380 DATA160,197,32,80,184,32,170,177
1390 DATA140,4,195,173,91,197,172,92
1400 DATA197,32,162,187,169,99,160,197
1410 DATA32,80,184,32,170,177,140,5,195
1420 DATA173,0,195,205,2,195,200,19,173
1430 DATA1,195,205,3,195,208,11,173,4
1440 DATA195,205,5,195,208,3,76,207,198
1450 DATA32,14,195,173,81,197,205,82
1460 DATA197,240,40,169,5,24,109,85,197
1470 DATA141,85,197,144,3,238,86,197
1480 DATA169,5,24,109,91,197,141,91,197
1490 DATA144,3,238,92,197,230,253,208,2
1500 DATA230,254,238,82,197,76,73,196
1510 DATA173,74,197,205,81,197,240,88
1520 DATA169,5,24,109,83,197,141,83,197
1530 DATA144,3,238,84,197,169,5,24,189
1540 DATA89,197,141,89,197,144,3,238,90
1550 DATA197,169,1,24,109,74,197,10,56
1560 DATA237,81,197,24,105,1,24,101,82
1570 DATA133,253,165,254,105,0,133,254
1580 DATA173,68,197,141,85,197,173,69
1590 DATA197,141,86,197,173,172,197,141
1600 DATA91,197,173,73,197,141,92,197
1610 DATA169,1,141,82,197,238,81,197,76
1620 DATA73,196,104,168,104,170,184,96
1630 DATA173,68,197,141,83,197,141,85
1640 DATA197,173,69,197,141,84,197,141
1650 DATA86,197,173,70,197,141,87,197
1660 DATA173,71,197,141,88,197,173,74
1670 DATA197,141,93,197,173,72,197,141
1680 DATA89,197,141,91,197,173,73,197
1690 DATA141,90,197,141,92,197,160,159
1700 DATA32,162,179,162,94,160,197,32
1710 DATA212,187,160,99,32,162,179,162
1720 DATA99,160,197,32,212,187,96
1730 DATA79160:REM"CHECKSUM"
1740 FORI=50536TO51123
1750 READX:POKEI,X:CC=CC+X
1760 NEXT
1770 READX:IFX<>CCTHENPRINT"ERROR EN SUMA CONTROL":STOP
1780 PRINT "HEXA II-ROT INSERTADO CORRECTAMENTE"
```

Listado assembly

```
1010 ;+++++++
```

```
2040              LDA   #<MEM1
2050              LDY   #>MEM1
2060              JSR   FSUB        ; FAC=MEM1-FAC
2070              LDX   #<MEM1
2080              LDY   #>MEM1
2090              JSR   MOVMF       ; MEM1=FAC
2110 ;++++ REALIZA MEM2=Y(I)*CS+X(I)*SN
2130              LDA   YILO
2140              LDY   YIHI
2150              JSR   MOVFM       ; FAC=Y(I)
2160              LDA   CSLO
2170              LDY   CSHI
2180              JSR   FMULT       ; FAC=Y(I)*CS
2190              LDX   #<MEM2
2200              LDY   #>MEM2
2210              JSR   MOVFM       ; MEM2=Y(I)*CS
2220              LDA   XILO
2230              LDY   XIHI
2240              JSR   MOVMF       ; FAC=X(I)
2250              LDA   SNLO
2260              LDY   SNHI
2270              JSR   FMULT       ; FAC=X(I)*SN
2280              LDA   #<MEM2
2290              LDY   #>MEM2
2300              JSR   FADD        ; FAC=MEM2+FAC
2310              LDX   #<MEM2
2320              LDY   #>MEM2
2330              JSR   MOVFM       ; MEM2=FAC
2350 ;++++ REALIZA X(I)=MEM1:Y(I)=MEM2
2370              LDA   #<MEM1
2380              LDY   #>MEM1
2390              JSR   MOVFM       ; FAC=MEM1
2400              LDX   XILO
2410              LDY   XIHI
2420              JSR   MOVMF       ; X(I)=FAC
2430              LDA   #<MEM2
2440              LDY   #>MEM2
2450              JSR   MOVFM       ; FAC=MEM2
2460              LDX   YILO
2470              LDY   YIHI
2480              JSR   MOVMF       ; Y(I)=FAC
2500 ;++++ FIN BUCLE??
2520              DEC   TEMPNP
2530              BEQ   CONTIN
2550 ;++++ INCREMENTA PUNTEROS TABLA
2570              LDA   #$05
2580              CLC
2590              ADC   XILO
2600              STA   XILO
2610              BCC   XNOHI
2620              INC   XIHI
2630 XNOHI
2640              LDA   #$05
2650              CLC
2660              ADC   YILO
2670              STA   YILO
2680              BCC   YNOHI
2690              INC   YIHI
2700 YNOHI
2710              JMP   START
2730 ;++++ BORRA/INIC PANTALLA
2750 CONTIN
2760              LDA   #$01
2770              STA   $C100
2780              STA   $C101
2790              STA   $C102
2800              JSR   $C10E       ;INIT HIRES
2820 ;++++ TRAZA LINEAS DESDE E%(I,J)
2840 ;  INICIALIZA VARIABLES
2860              JSR   SETUP
2870              LDA   #$01
2880              STA   IINDEX      ; I=1
2890              STA   JINDEX      ; J=1
2900              LDA   EBASLO      ; ALMACENA
2910              STA   ZPTEMP      ; PUNTERO
2920              LDA   EBASHI      ; E% EN
2930              STA   ZPTEMP+1    ; PAG. CERO
2950 ;  ALLA VAMOS - INICIO BUCLE I,J GIGANTE
2970 NEXTJ
2980              LDY   JINDEX
2990              LDA   (ZPTEMP),Y  ;GET E%(I,J)
3000              BNE   DOIT
3010              JMP   ONWARD
3030 ;++++ REALIZA X1%=X(I)+159
3040 ;      MLO=X1% LO:MHI=X1%HI
3060 DOIT
3070              LDA   XILO
3080              LDY   XIHI
3090              JSR   MOVFM       ; FAC=X(I)
3100              LDA   #<MEM1
3110              LDY   #>MEM1
3120              JSR   FADD        ; FAC=X(I)+159
3130              JSR   FLPINT      ; .Y/.A=INT (FAC)
3140              STY   MLO         ; X1% LO
3150              STA   MHI         ; X1% HI
3170 ;++++ REALIZA X2%=X(J)+159
3180 ;      NLO=X2% LO:NHI=X2% HI
3200              LDA   XJLO
3210              LDY   XJHI
3220              JSR   MOVFM       ; FAC=X(J)
3230              LDA   #<MEM1
3240              LDY   #>MEM1
3250              JSR   FADD        ; FAC=X(J)+159
3260              JSR   FLPINT      ; .Y/.A=INT(FAC)
3270              STY   NLO         ; X2% LO
3280              STA   NHI         ; X2% HI
3300 ;++++ REALIZA Y1%=99-Z(I)
3320              LDA   ZILO
3330              LDY   ZIHI
3340              JSR   MOVFM       ; FAC=Z(I)
3350              LDA   #<MEM2
3360              LDY   #>MEM2
3370              JSR   FSUB        ; FAC=99-Z(I)
3380              JSR   FLPINT      ; .Y/.A=INT(FAC)
3390              STY   Y1          ; Y1%
3410 ;++++ REALIZA Y2%=99-Z(J)
3430              LDA   ZJLO
3440              LDY   ZJHI
3450              JSR   MOVFM       ; FAC=Z(J)
3460              LDA   #<MEM2
3470              LDY   #>MEM2
3480              JSR   FSUB        ; FAC=99-Z(J)
3490              JSR   FLPINT      ; .Y/.A=INT(FAC)
3500              STY   Y2          ; Y2%
3520 ; PREPARA PARA LINESUB
3540 ; COMPARA DOS BYTES DE X1%=X2%
3560              LDA   MLO
3570              CMP   NLO
3580              BNE   NOPE
3590              LDA   MHI
3600              CMP   NHI
3610              BNE   NOPE
3630 ; COMPARA AHORA Y1=Y2
3650              LDA   Y1
3660              CMP   Y2
3670              BNE   NOPE
3690 ; ELUDE LINESUB SI SON IGUALES
3710              JMP   ONWARD
3720 NOPE
3730              JSR   LINSUB      ;TRAZA LINEA
3750 ; INCREMENTA PUNTEROS J
3770 ONWARD
3780              LDA   IINDEX      ;PRIMERA PRUEBA
3790              CMP   JINDEX      ;I=<I ?
3800              BEQ   NEXTI
3820 ; INCREMENTA PUNTEROS J
3840              LDA   #$05
3850              CLC
3860              ADC   XJLO
3870              STA   XJLO
3880              BCC   XJNOHI
3890              INC   XJHI
3900 XJNOHI
3920 ; INCREMENTA ZJLG/ZJHI
3940              LDA   #$05
3950              CLC
3960              ADC   ZJLO
3970              STA   ZJLO
3980              BCC   ZJNOHI
3990              INC   ZJHI
4000 ZJNOHI
4020 ; INCREMENTA 1 EL PUNTERO DE E%
4040              INC   ZPTEMP
4050              BNE   NOHIBY
4060              INC   ZPTEMP+1
4070 NOHIBY
4080              INC   JINDEX
4090              JMP   NEXTIJ
4110 ; INCREMENTA PUNTEROS I
4130 NEXTI
4140              LDA   NP          ;PRIMERA PRUEBA
4150              CMP   IINDEX      ;I=<NP ?
4160              BEQ   EXIT
4180 ; INCREMENTA XILO/XIHI
4200              LDA   #$05
4210              CLC
4220              ADC   XILO
4230              STA   XILO
4240              BCC   XINOHI
4250              INC   XIHI
4260 XINOHI
4280 ; INCREMENTA ZILO/ZIHI
4300              LDA   #$05
4310              CLC
4320              ADC   ZILO
4330              STA   ZILO
4340              BCC   ZINOHI
4350              INC   ZIHI
4360 ZINOHI
4380 ; INCREMENTA PUNTEROS DE E% (TRUCO!)
4400              LDA   #$01        ; CUANDO INC I
4410              CLC               ; SE AÑADIRA
4420              ADC   NP          ; 2*(NP+1)-(I-1)
4430              ASL   A           ; AL PUNTERO
4440              SEC               ; ZP DE E%
4450              SBC   IINDEX
4460              CLC
4470              ADC   #$01
4480              CLC
4490              ADC   ZPTEMP
4500              STA   ZPTEMP
4510              LDA   ZPTEMP+1
```

```
4520                ADC   #$00
4530                STA   ZPTEMP+1
4550 ; REINICIALIZA XJLO/XJHI
4570                LDA   XBASLO
4580                STA   XJLO
4590                LDA   XBASHI
4600                STA   XJHI
4620 ; REINICIALIZA ZJLO/ZJHI
4640                LDA   ZBASLO
4650                STA   ZJLO
4660                LDA   ZBASHI
4670                STA   ZJHI
4690 ; REINICIALIZA JINDEX
4710                LDA   #$01
4720                STA   JINDEX
4740 ; INCREMENTA IINDEX
4760                INC   IINDEX
4770                JMP   NEXTIJ
4790 ; EL BUCLE GIGANTE ACABA AQUI
4800 ;
4810 ;++++ RESTAURA LOS REGISTROS ++++
4820 ;
4830 EXIT
4840                PLA
4850                TAY
4860                PLA
4870                TAX
4880                PLA
4890                RTS
4930 ;++++ SUBRUTINA
4940 ;
4950 SETUP
4960                LDA   XBASLO
4970                STA   XILO
4980                STA   XJLO
4990                LDA   XBASHI
5000                STA   XIHI
5010                STA   XJHI
5020                LDA   YBASLO
5030                STA   YILO
5040                LDA   YBASHI
5050                STA   YIHI
5060                LDA   NP
5070                STA   TEMPNP
5080                LDA   ZBASLO
5090                STA   ZILO
5100                STA   ZJLO
5110                LDA   ZBASHI
5120                STA   ZIHI
5130                STA   ZJHI
5140                LDY   #159
5150                JSR   SNGFT       ; FAC=159
5160                LDX   #<MEM1
5170                LDY   #>MEM1
5180                JSR   MOVMF       ; MEM1=159 IN FPT
5190                LDY   #99
5200                JSR   SNGFT       ; FAC=99
5210                LDX   #<MEM2
5220                LDY   #>MEM2
5230                JSR   MOVMF       ; MEM2=99 IN FTP
5240                RTS
```

Programa de prueba en BASIC

```
1000 REM ** TRAZADO CUBOS ROTATORIOS **
1010 IFA=0THENA=1:LOAD"PLOTSUB.HEX",8,1
1020 IFA=1THENA=2:LOAD"LINESUB.HEX",8,1
1030 IFA=2THENA=3:LOAD"0-ROT.HEX",8,1
1040 PRINTCHR$(147)"ESPERA 17 SEGUNDOS"
1050 REM**DIMENSIONA LAS TABLAS**
1060 NP=24:REM NUMERO DE PUNTOS
1070 DIM X(NP),Y(NP),Z(NP)
1080 DIM E%(NP,NP):REM CONEXION DE LAS ESQUINAS
1090 REM**INICIALIZA TABLAS**
1100 REM--DATOS INICIALES COORDENADAS CUBO
1110 DATA   50,   50,   50:REM--------/1
1120 DATA  -50,   50,   50:REM TOP FOUR /2
1130 DATA  -50,  -50,   50:REM POINTS /3
1140 DATA   50,  -50,   50:REM--------/4
1150 DATA   50,   50,  -50:REM--------/5
1160 DATA  -50,   50,  -50:REM BOT FOUR /6
1170 DATA  -50,  -50,  -50:REM POINTS /7
1180 DATA   50,  -50,  -50:REM--------/8
```

La siguiente sección del programa lee (READ) los datos definidos anteriormente en las tablas X(), Y(), Z(). Para cubos minúsculos los valores DATA iniciales se reducirán mediante el coef. 0.3

```
1190 REM ** LEE DATOS DE CUBOS 1-2
1200 FORI=1TO8:REM CENTRO DEL CUBO GRANDE
1210 READX(I),Y(I),Z(I)
1220 NEXT
1230 RESTORE
1240 FORI=1TO8:REM CUBO PEQUEÑO DE LA DERECHA
1250 READX,Y,Z
1260 X(I)=.3*X+120:Y(I)=.3*Y:Z(I)=.3*Z
1270 NEXT
```

```
1280 RESTORE
1290 FORI=17TO24:REM CUBO PEQUEÑO IZQ
1300 READX,Y,Z
1310 X(I)=.3*X-120:Y(I)=.3*Y:Z(I)=.3*Z
1320 NEXT
```

La siguiente sección hace girar las coordenadas de los puntos equivalente a una perspectiva cambiada en 45°

```
1330 REM ** ROTACION ESPACIAL SOBRE EJE X **
1340 FORI=1TONP
1350 Y(I)=Y(I)*COS(π/4)-Z(I)*SIN(π/4)
1360 Z(I)=Z(I)*COS(π/4)+Y(I)*SIN(π/4)
1370 NEXT
1380 REM** ROTACION ESPACIAL DE π/4 SOBRE EL EJE Z
1390 FORI=1TONP
1400 X(I)=X(I)*COS(π/4)-Y(I)*SIN(π/4)
1410 Y(I)=Y(I)*COS(π/4)+X(I)*SIN(π/4)
1420 NEXT
```

La siguiente sección define los puntos que se han de conectar dentro de la figura en cada uno de los tres cubos, mediante la tabla de adyacencia E%(,)

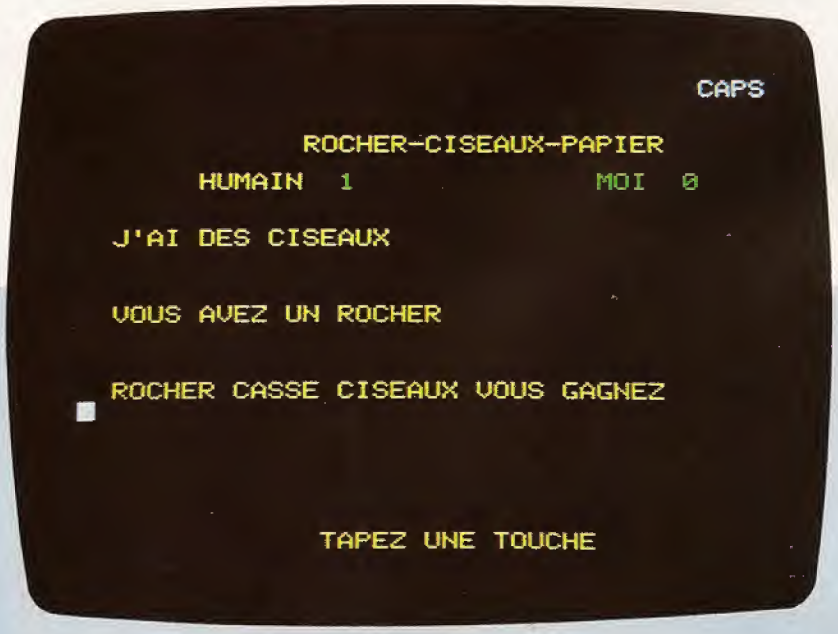
```
1430 REM ** DATOS CONEXION ESQUINAS **
1440 REM--CUBO TAMAÑO INTERMEDIO
1450 E%(1,2)=1:REM CONEXION DE 1 CON 2
1460 E%(2,3)=1:E=(3,4)=1:E%(4,1)=1
1470 E%(5,6)=1:REM CUADRADO BASE
1480 E%(6,7)=1:E%(7,8)=1:E%(8,5)=1
1490 E%(5,1)=1:REM ESQUINAS DE ARRIBA ABAJO
1500 E%(6,2)=1:E%(7,3)=1:E%(8,4)=1
1510 REM------------------------
1520 REM--CUBO PEQUEÑO DERECHA
1530 FORI=9TO16:FORJ=9TO16
1540 E%(I,J)=E%(I-8,J-8)
1550 NEXT:NEXT
1560 REM--LEFT SMALL CUBE
1570 FORI=17TO24:FORJ=17TO24
1580 E%(I,J)=E%(I-8,J-8)
1590 NEXT:NEXT
1600 REM**SIMETRIZA E%(I,J)**
1610 FORI=1TONP:FORJ=1TONP
1620 IFE%(I,J)<>0THENE%(J,I)=1
1630 NEXT:NEXT
```

En este punto el programa llama al código máquina para que ejecute los cálculos necesarios del giro y dibuje la nueva figura. Esta llamada se hace reiteradamente desde el interior de un bucle que hace girar los tres cubos 360° antes de terminar

```
1640 REM ************************************************
1650 REM ** TRAZADO CUBO ROTATORIO **
1660 SA=2*π/45:CS=COS(SA):SN=SIN(SA)
1670 GOSUB1790:REM INICIALIZACION
1680 FOR A=0 TO 2*π STEP SA
1690 SYS50536:REM ROTACION
1700 NEXTA:REM ANGULO SIGUIENTE
1710 GETA$:IFA$=""THEN1710
1720 REM ####################################
1730 GOSUB1750:REM RESTAURA PANTALLA
1740 END
1750 REM**RESTAURA PANTALLA**
1760 POKE49408,0:SYS49422
1770 PRINTCHR$(147)
1780 RETURN
1790 REM**INICIALIZA ROTSUB**
1800 REM************************************************
1810 REM**                NOTA--NO DEFINIR NUEVAS     **
1820 REM**                 VARIABLES ENTRE ESTA       **
1830 REM**               SUBRUTINA Y LA LLAMADA       **
1840 REM**                    AL C/M YA QUE           **
1850 REM**               CAMBIARIAN LAS TABLAS        **
1860 REM**                  DE DIRECCION BASE         **
1870 REM************************************************
1880 X(1)=X(1):REM X(1) VARIABLE EN CURSO
1890 POKE50500,PEEK(71):REM X(1)LO
1900 POKE50501,PEEK(72):REM X(1)HI
1910 Y(1)=Y(1):REM Y(1) VARIABLE EN CURSO
1920 POKE50502,PEEK(71):REM Y(1)LO
1930 POKE50503,PEEK(72):REM Y(1)HI
1940 Z(1)=Z(1):REM Z(1) VARIABLE EN CURSO
1950 POKE50504,PEEK(71)
1960 POKE50505,PEEK(72)
1970 POKE50506,NP:REM NUMERO DE PUNTOS
1980 CS=CS:REM HACE A CS VAR EN CURSO
1990 POKE50507,PEEK(71)
2000 POKE50508,PEEK(72)
2010 SN=SN:REM HACE A SN VAR EN CURSO
2020 POKE50509,PEEK(71)
2030 POKE50510,PEEK(72)
2040 E%(1,1)=E%(1,1):REM E% VARIABLE EN CURSO
2050 POKE50511,PEEK(71)
2060 POKE50512,PEEK(72)
2070 RETURN
```

Mano a mano

He aquí un interesante juego, conocido bajo la denominación "Piedra, papel y tijeras". Esta versión en BASIC ha sido escrita por Peter Shaw para el Oric



Tradicionalmente este juego se practica con las manos. Los dos jugadores, con sus manos a la espalda, deben mostrarlas simultáneamente imitando con ellas el objeto escogido. Deben elegir entre piedra, papel y tijeras. Las tijeras cortan el papel, la piedra rompe las tijeras, el papel cubre la piedra. Una vez hecha la elección, pulse C para tijeras, R para piedra, P para papel. Gana el jugador que totalice diez puntos.

```
10 REM PIEDRA, PAPEL, TIJERAS
20 CLS**PETER SHAW**
30 PAPER 0:INK 3
40 A=INT(RND(1)*3):PING
50 PLOT 12,2,"PIEDRA-PAPEL-TIJERAS"
55 PLOT 6,4,"HUMANO "+STR$(HS)
56 PLOT 29,4,"YO "+STR$(OS)
60 GET A$:ZAP:FOR P=0 TO 7:INK P:WAIT 4;
   NEXT P
65 CLS
70 IF A$="R" THEN V=0
80 IF A$="P" THEN V=1
90 IF A$="C" THEN V=2
100 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:
    PRINT
110 PRINT"YO TENGO ";
120 IF A=0 THEN PRINT"PIEDRA"
130 IF A=1 THEN PRINT"PAPEL"
140 IF A=2 THEN PRINT"TIJERAS"
150 PRINT:PRINT:PRINT
160 PRINT"TU TIENES ";
170 IF V=0 THEN PRINT"PIEDRA"
180 IF V=1 THEN PRINT"PAPEL"
190 IF V=2 THEN PRINT"TIJERAS"
200 PRINT:PRINT:PRINT:SHOOT
210 IF V=A THEN PRINT"EMPATE"
220 IF V=0 AND A=1 THEN PRINT"PAPEL
    CUBRE PIEDRA. YO GANO":OS=OS+1
230 IF V=0 AND A=2 THEN PRINT"PIEDRA ROMPE
    TIJERAS. TU GANAS":HS=
    HS+1
240 IF V=1 AND A=0 THEN PRINT"PAPEL
    CUBRE PIEDRA. TU GANAS":HS=
    HS+1
250 IF V=1 AND A=2 THEN PRINT"TIJERAS
    CORTAN PAPEL. TU GANAS":OS=OS+1
260 IF V=2 AND A=0 THEN PRINT "PIEDRA ROMPE
    TIJERAS. YO GANO":OS=OS+1
270 IF V=2 AND A=1 THEN PRINT"TIJERAS
    CORTAN PAPEL. TU GANAS":HS=HS+1
280 PLOT 13,23,"PULSA UNA TECLA"
290 EXPLODE:WAIT 30
295 IF HS=10 OR OS=10 THEN 310
300 GOTO 30
310 CLS
320 IF HS=10 THEN PING:PRINT"BRAVO.TU
    GANAS"
330 IF OS=10 THEN ZAP:PRINT"YO GANO OTRA
    VEZ"
340 PRINT:PRINT:PRINT
345 WAIT 20:EXPLODE
350 END
```

Scout y B-Star

Continuando con nuestro estudio de la planificación estratégica en AI, examinaremos algunas estrategias alternativas para juegos de azar

El procedimiento *alfa-beta*, que explicamos en el capítulo anterior, representa un gran avance sobre la minimaximización directa (puesto que identifica y suprime del árbol del juego las ramificaciones redundantes); durante muchos años ha sido el punto central de los programas de ajedrez por ordenador más eficaces. Pero recientemente se han propuesto dos estrategias alternativas. Una es el *algoritmo Scout*, de Judea Pearl; la otra, el *algoritmo B-Star (B*)*, de Hans Berliner.

La esencia del método Scout consiste en disponer de una función de evaluación sumamente afinada que se pueda utilizar para rechazar, sin más búsqueda, los movimientos poco plausibles. Sólo es necesario examinar en profundidad los movimientos que parecen más prometedores.

El método B* escudriña los movimientos del nivel superior del árbol e intenta, lo más rápidamente posible, cumplir uno de estos dos cometidos:

- Demostrar que el movimiento *aparentemente* mejor es en realidad el mejor disponible.
- Demostrar que ninguno de los movimientos alternativos es mejor que otro.

Esta estrategia gemela se implementa mediante un par de procedimientos denominados ProveBest (demostrar mejor) y RefuteBest (refutar mejor), que se basan en la asignación de los valores a cada nudo del árbol: uno, una evaluación optimista; otro, una pesimista. El objetivo es obligar al algoritmo de búsqueda a concentrarse en áreas del árbol del juego en donde hay incertidumbre y donde esta falta de certeza podría incidir en la decisión final.

Sería equivocado suponer que la búsqueda arborescente es el único enfoque a los juegos por ordenador. Existen algunos juegos interesantes en los que la filosofía de búsqueda parece no dar buenos resultados, independientemente de los ingeniosos trucos que se adopten para perfilar el proceso. Entre estos se incluyen muchos de los juegos de naipes más populares (en particular el bridge y el poker) y varios juegos de tablero (como el *go* y el *go-moku*).

Estos juegos se pueden practicar con diversos niveles de destreza, y se dice merecidamente que son los más sobresalientes exponentes de juegos inteligentes. Pero todos los intentos por programarlos de acuerdo al esquema de la búsqueda arborescente han tropezado con obstáculos inesperados. Una

Búsqueda vana
Aunque los ordenadores pueden jugar inteligentemente a algunos juegos empleando métodos de búsqueda para anticiparse a un cierto número de movimientos, hay muchos juegos en los que las estrategias de búsqueda no son eficaces; ello se debe a que el juego posee un elemento de azar, como el *backgammon*, o bien a que el árbol del juego se bifurca rápidamente en sucesivos movimientos para producir una ingente cantidad de posibles permutaciones de movimientos. Para crear programas que puedan jugar a esta clase de juegos, se deben procurar métodos alternativos

David Lawrence

Terminología de la búsqueda arborescente

Árbol de juego	Estructura arborescente que se forma considerando los movimientos posibles, seguidos por las posibles réplicas del adversario, etc.
Nivel	Un nivel del árbol de juego.
Anticipación	Proceso de construcción de un árbol de juego.
Valor de un nudo	El valor asignado a un nudo del árbol de juego mediante el examen de los valores por debajo de él, trabajando desde abajo.
Minimaximización	Elección del valor que sustentará el árbol mediante la minimización en los niveles impares (mueve el otro) y la maximización en los pares (mueve uno).
Algoritmo "alfa-beta"	Refinamiento de la minimaximización que elimina las porciones del árbol de juego que se estima no pueden tener incidencia en el nivel superior.
Factor de bifurcación	Cantidad media de bifurcaciones o movimientos en cada nivel del árbol de juego. El *go*, juego oriental, posee un factor de bifurcación de más de 200.

razón es que el factor de bifurcación es demasiado amplio, generando una cantidad tan enorme de posibles combinaciones de movimientos que el ordenador no puede manipularlas todas al mismo tiempo.

Una respuesta más profunda es que los algoritmos de búsqueda arborescente son una burda aproximación a la forma en que el jugador humano aborda el problema: los expertos buscan sólo mentalmente en sus propios árboles de juego, construidos internamente en circunstancias especiales, e incluso así sin demasiada eficacia.

Hans Berliner, quien ideó el método B*, desarrolló un programa de *backgammon* que en 1980, en una partida de desafío, venció al campeón del mundo. Pero el programa no efectúa ninguna búsqueda en absoluto, al menos en el sentido convencional. Si usted piensa en cómo se juega una partida de *backgammon*, comprenderá que su árbol de juego es "probabilístico": el papel de los dados introduce bifurcaciones que están bajo el control de cualquiera de los jugadores, lo que dificulta la implementación de muchos de los procedimientos de anticipación esbozados previamente.

En lo que el programa de Berliner es muy bueno (mucho mejor que nosotros) es en calcular las probabilidades de que se produzcan durante el juego diversas combinaciones de dados y fichas. Posee, asimismo, una función de evaluación muy sofisticada. De hecho, en realidad emplea varias funciones para diferentes fases del juego y varía constantemente entre ellas a medida que el juego va progresando.

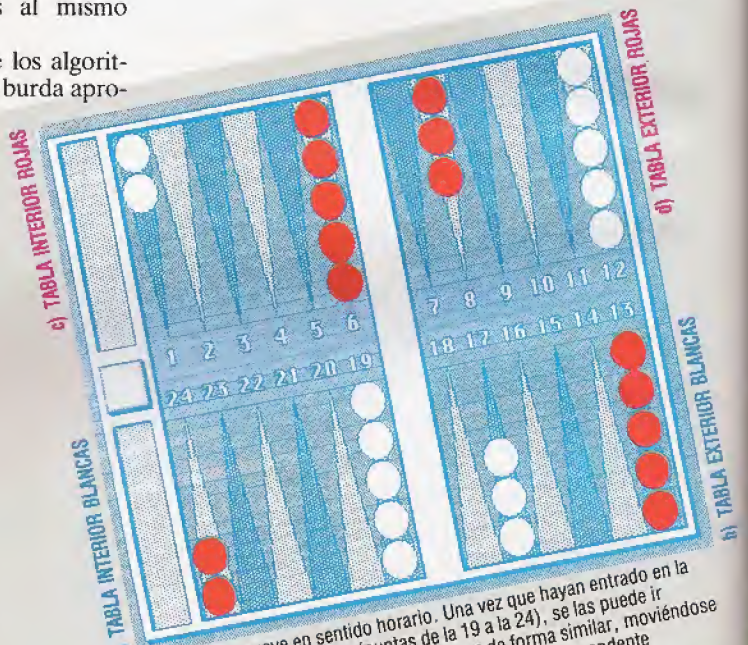
El *go* es un juego oriental, que se juega desplazando piezas sobre una cuadrícula de 18 por 18, con el objeto de rodear áreas de la cuadrícula para ganar territorio, y de rodear las piezas del oponente para eliminarlas del juego. Carece de un elemento de azar, pero el factor de bifurcación es tan inmenso que las técnicas basadas en búsqueda resultan infructuosas.

Los mejores programas de *go* perciben el tablero en términos de unidades mayores que meras piezas individuales (tales como *cadenas* y *ejércitos*) que son agrupamientos significativos para el ojo humano; una de las razones por las cuales la programación del *go* está más atrasada que la del ajedrez puede que sea que nuestra comprensión de la percepción humana es inadecuada. Quizá los japoneses, que veneran el juego, consideren al *go* como un proyecto adecuado para sus máquinas paralelas de la quinta generación. Ciertamente, un programa que juegue al *go* con éxito desarrollará el concepto de inteligencia artificial hasta sus límites.

Ya existen programas de *go* para los actuales ordenadores personales, y próximamente consideraremos los problemas que entraña la programación del juego. Estos programas, sin embargo, no tienen posibilidades de derrotar a los maestros del juego.

Dentro de la tabla interior

El *backgammon* se juega sobre un tablero compuesto por 24 "puntas" triangulares en las que se mueven las piezas, en función del lanzamiento de un par de dados. El objetivo del juego es desplazar todas las piezas propias alrededor del tablero y colocarlas en el final antes que el oponente. Si los números de los dados son diferentes, uno puede mover una pieza la cantidad total de puntos que indican los dados, o bien mover dos piezas por separado según el número individual de cada dado. Cuando se mueve una pieza de acuerdo a la suma de los dados, se considera que está efectuando dos movimientos separados y se dice que "captura" en la punta fronteriza entre los dos movimientos. Las puntas ocupadas por dos o más piezas de un color no se pueden mover ni "capturar" por el lado oponente. Si una pieza ocupa una punta y es "capturada" por una pieza oponente, se la elimina del tablero y su dueño la debe volver a introducir antes de poder efectuar otro movimiento. La ilustración muestra un movimiento de la partida entre el programa de *backgammon* de Hans Berliner y el campeón del mundo, Luigi Villa, que se celebró en Montecarlo en 1980



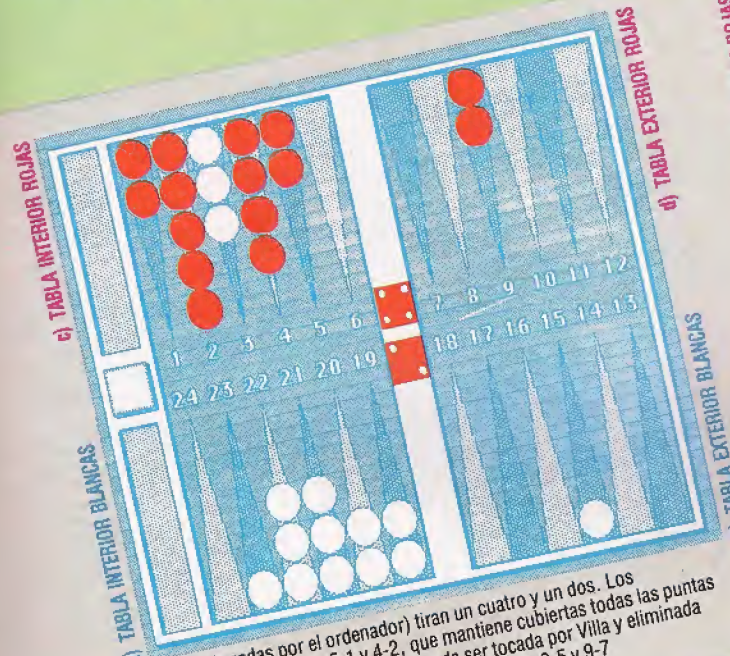
Las blancas se mueve en sentido horario. Una vez que hayan entrado en la "table interior" de las blancas (puntas de la 19 a la 24), se las puede ir eliminando progresivamente. Las rojas juegan de forma similar, moviéndose en sentido antihorario, siguiendo las puntas por orden descendente

Para ilustrar los importantes conceptos de la búsqueda arborescente, ideamos un juego artificial que era casi una búsqueda pura. Este juego lo presentamos en el capítulo anterior en versión de BASIC BBC, haciendo un uso intensivo de las funciones recursivas con parámetros.

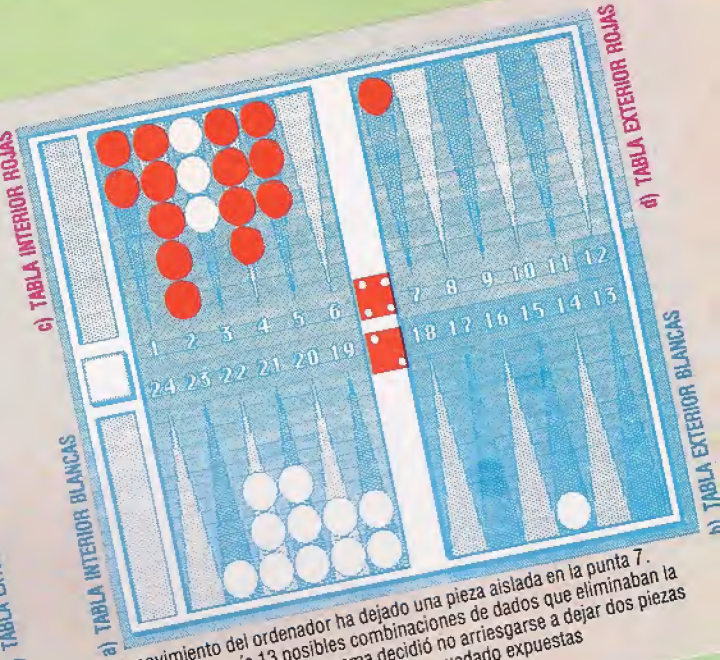
El juego permite que el jugador y el ordenador se turnen para seleccionar una de cuatro funciones que modifican un valor existente para producir uno nuevo. El jugador intenta reducir el valor a −255, mientras que la máquina intenta hacer que exceda de 255. En cada etapa el ordenador utiliza la minimaximización *alfa-beta* para seleccionar cuál de las cuatro funciones definidas es mejor para sus fines.

Teniendo presentes a los usuarios del Commodore 64 y el Spectrum, cuyos BASIC carecen de parámetros y variables locales, repetimos el programa en una versión que sólo utiliza GOSUBs. Los parámetros y las variables locales (a excepción de D, el contador de profundidad) se han sustituido por matrices, DIMensionadas en la línea 1100. Ahora D actuará como una especie de puntero de pila, llevando el registro de los elementos de las matrices a los que se esté accediendo en cada momento.

Estas dos rutinas se diferencian fundamentalmente en que todo ha de ir subindexado por D, en particular la matriz A(), utilizada para retener el mejor valor hallado hasta ahora, y B(), empleada para retener el peor. Ello asegura que los valores utilizados de *alfa* y *beta* se aplicarán a los niveles correctos del árbol.



Las rojas (jugadas por el ordenador) tiran un cuatro y un dos. Los movimientos obvios son 5-1 y 4-2, que mantiene cubiertas todas las puntas y no deja ninguna pieza aislada que pueda ser tocada por Villa y eliminada del tablero. En cambio, el ordenador decidió mover 9-5 y 9-7



El movimiento del ordenador ha dejado una pieza aislada en la punta 7. Aunque Villa tenía 13 posibles combinaciones de dados que eliminaban la pieza del ordenador, el programa decidió no arriesgarse a dejar dos piezas en la punta 9 que, al moverlas, hubieran quedado expuestas

El juego de los números

```
 80 GOSUB 1000:REM INICIALIZACION
 90 GOSUB 1600:REM INSTRUCCIONES
100 :
110 REM **** BUCLE PRINCIPAL DEL PROGRAMA ****
120 GOSUB 2000:REM PREPARAR NUEVO JUEGO
130 INPUT"QUIEN JUEGA PRIMERO (1=TU,2=YO)";H1
140 IF H1<1 OR H1>2 THEN 130
150 REM *** BUCLE DEL JUEGO ***
160 IF H1=1 THEN GOSUB 3000
170 REM ** TURNO DE LA PERSONA **
180 GOSUB 3500:REM VISUALIZACION DEL TABLERO
190 H1=1:REM SIEMPRE 1 TRAS EL 1ER CICLO
200 GOSUB 4000: REM COMPROBAR GANADOR
210 IF EG=0 THEN GOSUB 5000
220 REM ** TURNO DEL ORDENADOR **
230 GOSUB 3500:REM VISUALIZAR ESTADO DEL JUEGO
240 GOSUB 4000: REM COMPROBAR SI FINAL DEL JUEGO
250 IF EG=0 AND M<=33 THEN 150:REM BUCLE ATRAS
260 REM ***** FINAL ****
270 GOSUB 6000: REM FELICITACIONES
280 INPUT"OTRA PARTIDA (1=SI,2=NO)";Y
290 IF Y<>2 THEN 280
300 IF Y=1 THEN 110: REM NUEVA PARTIDA
310 PRINT"HASTA OTRA Y GRACIAS POR JUGAR"
320 END
330 :
340 REM ***** MAXIMIZAR ****
350 D=D+1:C2=C1+1
360 IF D>=MD OR ABS(V(D))>HI THEN A(D)=V(D):D=D-1:RETURN
370 REM ** SI NO PROFUNDIZAR MAS **
380 P(D)=0
390 REM ** A TRAVES DEL ARBOL **
400 P(D)=P(D)+1:H=P(D):V=V(D):GOSUB 5500:REM REALIZAR MOVIMIENTO
410 IF D=1 THEN PRINT CHR$(64+H);"=";
420 D1=D+1
430 A(D1)=A(D):B(D1)=B(D):V(D1)=V:GOSUB 700:REM LLAMAR MINIMIZAR
600 IF B(D+1)>A(D) THEN A(D)=B(D+1):K(D)=P(D)
610 IF D=1 THEN PRINT B(D+1);":";
620 IF P(D)<=3 AND A(D)<B(D) THEN 550
630 IF D=1 THEN BV=A(D):HH=K(D):REM GUARDAR MEJOR HASTA AHORA
640 D=D-1:RETURN
650 :
700 REM **** MINIMIZAR ****
710 D=D+1:C2=C2+1
720 IF D>=MD OR ABS (V(D))>HI THEN B(D)=V(D):D=D-1:RETURN
730 (P(D)=0
740 REM ** A TRAVES DEL ARBOL **
750 P(D)=P(D)+1:H=P(D):V=V(D):GOSUB 5500:REM EFECTUAR MOVIMIENTO
760 D1=D+1:A(D1)=A(D):B(D1)=B(D):V(D1)=V
770 GOSUB 500:REM LLAMAR MAXIMIZAR
780 IF A(D+1)<B(D) THEN B(D)=A(D+1)
790 IF P(D)<=3 AND B(D)>A(D) THEN 740
800 D=D-1:RETURN
810 :
1000 REM **** INICIALIZAR ****
1010 BL$=" "
1020 REM ** DEFINICION DE LAS CUATRO FUNCIONES **
1030 DEF FNA(X)=2*X-7
1040 DEF FNB(X)=INT(X/2)+1
1050 DEF FNC(X)=4*X+17
1060 DEF FND(X)=3*X-4
1070 LO=-255:HI=255
1080 REM ** MATRICES QUE UTILIZA MINIMAX **
1090 D=16
1100 DIM V(D),A(D),B(D),P(D),K(D)
1110 RETURN
1120 :
1600 REM **** INSTRUCCIONES **
1610 PRINT"BIENVENIDO AL JUEGO DE LOS NUMEROS"
1620 PRINT"YO INTENTARE MAXIMIZAR, TU TAREA"
1630 PRINT"ES MINIMIZAR"
1640 PRINT"PARA VER EL EFECTO DE UN MOVIMIENTO PULSA:"
1650 PRINT"A, B, C O D. PULSA X PARA EFECTUARLO."
1660 PRINT:RETURN
1670 :
2000 REM **** PREPARACION ****
2010 M=0:V=INT(RND(1)*15)-8:REM ESTADO INICIAL
2020 EG=0
2030 PRINT"ESTADO INICIAL=";V
2040 RETURN
2050 :
3000 REM **** MOVIMIENTO DE LA PERSONA ****
3010 M=M+1:PRINT
3020 PRINT"TU MOVIMIENTO ES";
3030 INPUT H$
3040 IF H$="A" THEN PRINT FNA(V):H=1
3050 IF H$="B" THEN PRINT FNB(V):H=2
3060 IF H$="C" THEN PRINT FNC(V):H=3
3070 IF H$="D" THEN PRINT FND(V):H=4
3080 IF H$<>"X" THEN 3020:REM AUN NO SELECCIONADO EL MOVIMIENTO
3090 GOSUB 5500:REM EFECTUAR EL MOVIMIENTO
3100 RETURN
3110 :
3500 REM **** VISUALIZACION TABLERO ****
3510 PRINT:PRINT"          MOVIMIENTO";M;"-->";
3520 IF M<1 THEN RETURN
3530 PRINT CHR$(64+H);
3540 PRINT"=";V:PRINT:RETURN
3550 :
4000 REM **** PRUEBA GANADOR ****
4010 IF M<1 THEN RETURN
4020 EG=0
4030 IF V<LO THEN EG=-1
4040 IF V>HI THEN EG=1
4050 RETURN
4060 :
5000 REM **** MOVIMIENTO DEL ORDENADOR ****
5010 W=V:REM GUARDAR ESTADO ACTUAL
5020 M=M+1
5030 MD=6:REM MAX PROFUNDIDAD
5040 IF M<4 THEN MD=4
5050 IF M>8 THEN MD=8
5060 GOSUB 5200:REM -->H
5070 V=W:REM RESTAURAR ESTADO
5080 GOSUB 5500:REM EFECTUAR MOVIMIENTO
5090 RETURN
5100 :
5200 REM **** SELECCION MOVIMIENTO ****
5210 BV=LO:D=0
5220 V(1)=V:A(1)=LO:B(1)=HI
5230 GOSUB 500:REM MAXIMIZAR
5240 H=HH
5250 PRINT:INPUT"PULSA RETURN PARA CONTINUAR";Q
5260 RETURN
5270 :
5500 REM **** EFECTUAR UN MOVIMIENTO ****
5510 IF H=1 THEN V=FNA(V):RETURN
5520 IF H=2 THEN V=FNB(V):RETURN
5530 IF H=3 THEN V=FNC(V):RETURN
5540 IF H=4 THEN V=FND(V):RETURN
5550 :
6000 REM **** FELICITACIONES ****
6010 PRINT:PRINT"JUEGO TERMINADO"
6020 IF EG>0 THEN PRINT"HE GANADO YO"
6030 IF EG<0 THEN PRINT"HAS GANADO TU"
6040 IF EG=0 THEN PRINT"HA SIDO EMPATE"
6050 RETURN
```

Instrucciones transitorias

El CP/M cuenta con una gama de instrucciones que se cargan desde disco

Las instrucciones "transitorias" permiten que el usuario se comunique con las unidades de disco, administre archivos y controle otros dispositivos periféricos. Tras el encendido, el ordenador ejecutará una serie de comprobaciones para asegurarse de que estén presentes todos los componentes del sistema y que éstos sean correctos. Asimismo, el procesador abrirá canales a todos los periféricos que estén conectados. El ordenador enviará códigos a los diversos periféricos, en parte para comprobar que estén funcionando y, en parte, para prepararlos para recibir datos. Todo este proceso se conoce como *inicialización.*

Al inicializar algunas interfaces de periféricos, el ordenador espera que el periférico le devuelva un mensaje antes de proseguir con cualquier otra acción, tal como activar la ROM de BASIC. Un ejemplo de esto se puede apreciar cuando se instala un cartucho de juegos que, en efecto, se hace cargo del sistema operativo del ordenador y efectúa toda las entradas y salidas con el propio procesador, sin remitirse a la ROM de BASIC. De modo similar, cuando un ordenador compatible con CP/M abre un canal para su unidad de disco, espera a que la unidad le devuelva un mensaje.

El resultado de la inicialización del ordenador sobre la unidad de disco es que se obliga al cabezal de lectura/escritura del dispositivo a leer la primera pista del disco de sistema CP/M. Si no hay ningún disco presente, la unidad continuará girando hasta que se inserte un disco. La pista cero contiene el programa *cargador básico (boodstrap loader)*, que proporciona al ordenador las instrucciones necesarias para la carga en memoria del resto del programa CP/M. Si en la pista cero no hay ningún programa cargador, el ordenador generará un error DOS.

Después de que se haya terminado de cargar el CP/M en el ordenador, aparecerá en la pantalla un cursor intermitente junto a un símbolo A> (aunque algunas máquinas utilizan OA>). Se dice que éste es un *aviso* e indica que el sistema está listo para recibir una instrucción. También informa de cuál es la unidad de disco CP/M que está en ese momento en uso y en la que espera leer y escribir información. Se dice que la unidad a la cual está accediendo el CP/M en el momento es la *unidad actualmente conectada (logged).* En este caso, dado que apenas acabamos de cargar CP/M, todavía estamos en la unidad A.

Esto está muy bien si sólo poseemos una unidad de disco o bien si deseamos usar una sola. No obstante, si contamos con dos unidades de disco y queremos utilizar ambas, hemos de indicar al CP/M que busque la unidad B. Esto se consigue mediante la instrucción B y pulsando RETURN. El CP/M comprobará entonces que esté presente la unidad B y si en la misma hay en este momento un disco. De ser así, aparecerá en la pantalla el aviso B>, indicando que estamos ahora en la unidad B. Pero no estamos limitados a dos unidades. El CP/M puede acceder hasta a cuatro, aludiéndose a las otras como C y D, respectivamente.

Si bien no es muy usual que una misma máquina tenga instaladas cuatro unidades de disco separadas, muchos ordenadores, como el 380Z y el 480Z de Research Machines, poseen unidades de doble cara. En tales casos, lo normal es llamar A y B a las caras superiores de las unidades y C y D a las inferiores.

Tras conectar con el CP/M, lo normal es averiguar qué archivos están disponibles. Para hacer esto, debemos examinar el directorio del disco entrando la instrucción DIR o dir (el CP/M no diferencia entre mayúsculas o minúsculas). Ello hará que se visualice una lista de archivos, incluyendo todas las instrucciones que se pueden cargar y ejecutar desde CP/M.

Quizá parezca que tener que cargar y ejecutar un "archivo de instrucciones" con el mero fin de, por ejemplo, obtener información de estado sobre un archivo retenido en disco, es un proceso innecesariamente largo. ¿Por qué no podrían las instrucciones cargarse en RAM tras el encendido, y ejecutarse directamente cuando ello fuera necesario? La razón principal de este método, aparte de la obvia de ahorrar espacio de memoria, es facilitar la compatibilidad entre sistemas diferentes.

Como puede imaginar, el CP/M contiene instrucciones que nos permiten manipular archivos retenidos en disco. Si bien las aplicaciones CP/M han de ser estándares con el objeto de que sean portables de una máquina a otra, en el transcurso de los años muchos fabricantes han adaptado el sistema de acuerdo a las necesidades de sus propios ordenadores. Por ejemplo, en CP/M no hay ninguna instrucción de formateo de disco estándar, de modo que cada fabricante le añade a la lista de instrucciones transitorias del CP/M esta función en particular, por lo que muchos sistemas trabajan de forma diferente.

Como vimos en el capítulo anterior, el nombre de un archivo CP/M se compone de un nombre primario, un punto y una extensión. Esta última afecta a la forma en que se carga el archivo en el ordenador. En los discos de sistema CP/M hay un cierto número de archivos de instrucciones que contienen las instrucciones transito-

Extensión	Explicación	Ejemplo
ASM	Se requiere para arch. fuente de leng. máq.	**CODEPROG.ASM**
BAK	Copia de un arch. de texto creado mediante el editor	**MEMO.BAK**
BAS	Indica un archivo fuente en BASIC	**PROG.BAS**
COM	Extensión necesaria para un arch. trans. o de instrucciones	**PIP.COM**
HEX	Asignado a un arch. hexadecimal a nivel máq.	**GRAPHIC.HEX**
INT	Otorgado a un programa en BASIC compilado	**GAMES.INT**
PRN	Necesaria para producir listados de progs. en leng. assembly	**CODEPROG.PRN**
SUB	Arch. utilizado para ejecutar instrs. en lotes *(batch)*	**SPOOLER.SUB**
$$$	Archivo temporal creado mediante el editor	**PHONE.$$$**

mas del CP/M, llevando todas ellas la extensión .COM. Ello significa que cuando se carguen en el ordenador se ejecutarán automáticamente.

Al objeto de cargar un archivo de instrucciones, usted simplemente digita el nombre primario, pulsa RETURN y se ejecutará el programa de instrucciones. Otros archivos con diferentes extensiones se cargan también de forma diferente, e iremos viéndolos con posterioridad.

Anteriormente hemos examinado el uso de la instrucción DIRectorio. En CP/M hay una instrucción transitoria asociada llamada STAT, que proporciona información adicional sobre el disco y los archivos en él retenidos. Al ejecutarse, STAT visualizará la cantidad de memoria libre disponible para nuevos archivos. La instrucción visualizará, asimismo, cierta información relativa al disco, como cuánta memoria queda todavía disponible para otros archivos y qué clase de facilidades de lectura/escritura se permiten. Por ejemplo, el mensaje R/W significa que se puede leer el disco y escribir en él, mientras que R/O significa que el disco sólo se puede leer o, mejor dicho, que está protegido contra la escritura.

Además, la instrucción STAT también puede "bloquear" un disco de modo que sólo pueda ser leído. Esto se consigue digitando STAT D:=R/O, donde D alude al carácter DRIVE. Todo intento ulterior de escribir en este disco concreto provocará un mensaje BDOS. La instrucción también visualizará las extensiones de nombre de archivo y la cantidad de sectores de archivo lógicos utilizados por cada registro, y mediante el empleo de STAT es posible examinar el tamaño de archivos individuales simplemente digitando STAT seguido del nombre del archivo. Asimismo, STAT se puede usar para examinar y, si así se requiere, modificar el estado de los periféricos que estén instalados. A modo de ejemplo, la instrucción STAT DEV visualizará una lista de todos los dispositivos de entrada/salida (incluyendo la pantalla) que estén conectados.

Una de las características más importantes de un sistema operativo de disco es la capacidad de transferir archivos de un disco a otro. En CP/M, esta operación la lleva a cabo PIP (*Peripheral Interchange Program*: programa de intercambio de periféricos). Esta instrucción no hace otra cosa que copiar de disco a disco; asimismo, permite la salida de archivos por impresora o por otros dispositivos de entrada/salida.

Para poder utilizar PIP se debe cargar y ejecutar la instrucción digitando PIP y pulsando RETURN. Observe que ahora el aviso pasa a ser un asterisco, indicando que se está ejecutando PIP y que se está esperando la instrucción siguiente. El formato para la utilización de PIP es: D:COPYNAME = D:SOURCENAME. A modo de ejemplo, vamos a suponer que tenemos un archivo de texto HCAC.TXT en un disco y queremos copiarlo en un segundo disco bajo el nombre WORK.TXT. Debemos sacar el disco de sistema de la unidad y colocar el disco que contenga HCAC.TXT en la unidad A, y colocar el segundo disco en la unidad B. De modo que para copiar HCAC.TXT de la unidad A a la unidad B, el formato es: B:WORK.TXT=A:HCAC.TXT.

Observe que cuando se utiliza PIP la unidad objeto se coloca primero en la instrucción, antes de la unidad fuente, que va a continuación, y que cuando se alude a archivos que no son instrucciones debe incluirse el nombre completo, incluyendo la extensión. Tras haber completado esta operación, usted puede constatar si el archivo se ha copiado correctamente examinando el directorio de la unidad B.

Aunque tenga una sola unidad de disco también puede utilizar PIP, pero las unidades fuente y de destino serán, por supuesto, ambas A. El CP/M copiará sectores del archivo en la memoria y le informará cuándo se deben cambiar los discos de modo que se puedan volcar los sectores desde la memoria al disco de destino.

En caso de que deseáramos enviar HCAC.TXT a la impresora, utilizaríamos una forma similar de PIP. En este caso, impartimos la instrucción: PIP LPT:=B:HCAC.TXT. Hay varios puntos a destacar en esta instrucción, el primero de los cuales es el hecho de que PIP no se cargará separadamente del resto de la instrucción. Esto es posible en CP/M puesto que usted no tiene necesariamente que esperar a que PIP se cargue antes de que pueda ocurrir alguna otra cosa.

En segundo lugar, el dispositivo de destino no es un canal de unidad sino una impresora, que es la razón por la cual utilizamos la instrucción LPT (*Line PrinTer*). En esta instrucción, hemos incluido asimismo la letra de unidad que indica dónde hallar el archivo HCAC.TXT. Si la unidad B estuviera conectada en ese momento, esta inclusión no sería necesaria. En realidad, bajo algunas versiones de CP/M no es necesaria ni aunque estemos en la unidad A. Esto se debe a que el CP/M buscará en todas las unidades de disco disponibles antes de generar un error de "archivo no hallado".

Al transferir un archivo de un disco a otro también cambiamos el nombre del archivo. Ésta no es una exigencia de PIP, dado que un archivo se puede copiar y conservar el mismo nombre. Sin embargo, supongamos que deseáramos cambiar el nombre de un archivo en un disco sin tener que copiarlo. En este caso, utilizaríamos la instrucción REN (de *rename*), de modo que si deseáramos cambiar el nombre de WORK.TXT por HCAC.TXT, emplearíamos la instrucción REN HCAC.TXT=WORK.TXT, recordando que siempre se coloca primero el nombre objeto. Ello creará un nuevo nombre de archivo en el directorio y borrará el anterior.

Del mismo modo, cuando deseamos borrar un archivo de un disco, digitamos ERA seguido por el nombre del archivo. Como con todas las instrucciones destructivas de esta clase, ésta debe ser usada con sumo cuidado. El CP/M debe saber a qué disco se está refiriendo usted, de lo contrario quizá acabe borrando una versión equivocada del archivo. De modo que, aunque no es necesario, es un buen hábito incluir siempre el nombre de la unidad de la instrucción; por ejemplo, ERA B:HCAC.TXT.

El ASM

El CP/M, tal como se implementa en la mayoría de las máquinas, incluye su propia utilidad de ensamblador denominada ASM. Aunque con frecuencia los programadores lo ignoran, se trata de un potente ensamblador con capacidad en ensamblaje condicional, así como todas las directivas de ensamblador estándares. El único inconveniente de la utilización del ASM es que originalmente se desarrolló para usar con el ensamblador 8080, que luego se reemplazaría por el Z80. Si bien el código objeto generado por el 8080 se ejecutará en el Z80, usted encontrará que un ensamblador 8080 tal como el ASM no siempre aceptará los mnemotécnicos estándares del Z80. En el libro *Programming the Z80*, de Rodnay Zaks, hallará una relación completa de las diferencias. No obstante, si pretende utilizar regularmente el ASM en una máquina Z80, le aconsejamos que se consulte con Digital Research, que puede proporcionarle un juego de macrorrutinas para ejecutar con ASM

Trazado de memoria

Finalizada la construcción del trazador digital, nos concentraremos en su calibrado y en el desarrollo del software para controlarlo

El primer paso para desarrollar software para el trazador consiste en calibrar el hardware. La ejecución del programa *Calibrado* que ofrecemos aquí nos proporciona cuatro valores digitales de potenciómetro que corresponden a cuatro posiciones críticas del brazo: posición cero del brazo 1, posición cero del brazo 2, posición en 90° del brazo 1 y posición en 90° del brazo 2. Estos valores deben entrarse en el programa del trazador digital en las líneas 1200, 1210, 1240 y 1250, como parte del procedimiento definir—parametros. Los otros dos parámetros que pueden variar de un trazador a otro son las longitudes de los brazos, y éstas deben entrarse en el programa en las líneas 1220 y 1260. Observe que las longitudes de los brazos deben estar expresadas en milímetros y que el brazo 2 incluye la distancia desde el extremo del brazo hasta el cruce de los ejes.

Habiendo obtenido estos valores de calibrado, podemos utilizarlos para convertir en ángulos las lecturas digitales proporcionadas por los potenciómetros (a través del convertidor de analógico a digital del propio BBC Micro). Esta información angular se puede convertir en desplazamientos horizontales y verticales de la mira desde el punto de montaje del brazo. En general, un ángulo se puede calcular aplicando la siguiente fórmula:

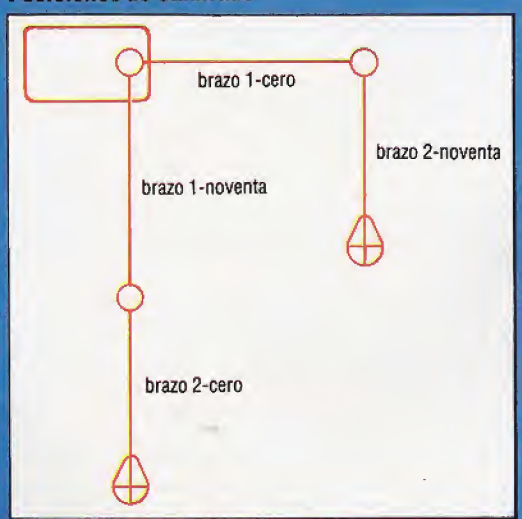
angulo=(ADVAL(n)−brazocero)*90/(brazonoventa-brazocero)

Puesto que la segunda parte de esta expresión es una constante, es mejor calcular el valor al comien-

Calibrado

Coloque el tablero del trazador hacia arriba de modo que la caja plástica quede en el rincón superior izquierdo. Para poder calibrar con total éxito el trazador necesitamos 4 informaciones: la posición en 0° del brazo 1, la posición en 90° del brazo 1, la posición en 0° del brazo 2 y la posición en 90° del brazo 2. Mientras se ejecuta el programa de calibrado, desplace el brazo del trazador de modo que el brazo 1 quede paralelo respecto al margen superior del tablero y el brazo 2, perpendicular. Apunte en un papel los valores digitales del brazo 1 y el brazo 2. Mueva el ensamblado del brazo de modo que ahora los brazos 1 y 2 estén en línea y paralelos al borde izquierdo del tablero. Vuelva a anotar las lecturas para los brazos 1 y 2. Pulse una tecla del teclado para pasar a la modalidad de ángulo. Entre las 4 lecturas que ha apuntado en respuesta a las preguntas y después compruebe que los ángulos visualizados para ambos brazos sean correctos. De no ser así, repita este proceso hasta que lo sean. Conserve la nota donde apuntó los 4 valores digitales del brazo, porque serán necesarios posteriormente.

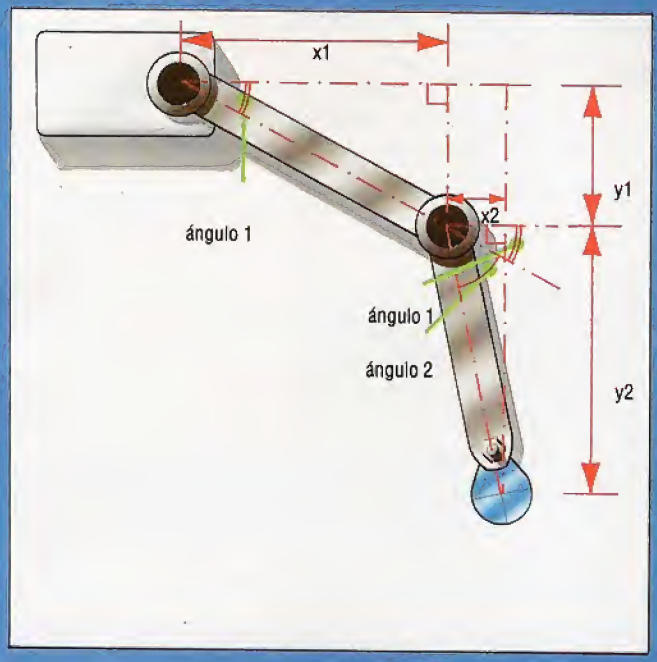
Posiciones de calibrado




```
1000 REM **** CALIBRADO TRAZADOR BBC ****
1010 PROCcalibrar
1020 PROCvisualizar_angulo
1030 END
1040 DEF PROCcalibrar
1050 CLS
1060 PRINT TAB(5,10);"ANGULO 1";TAB(20);"ANGULO 2"
1070 REPEAT
1080 PRINT TAB(5,12);SPC(40)
1090 PRINT TAB(5,12);ADVAL(1);TAB(20);ADVAL(2)
1100 PROCdemora(600)
1110 A$=INKEY$(1)
1120 UNTIL A$<>""
1130 ENDPROC
1150 DEF PROCvisualizar_angulo
1160 PROCdefinir_angulos
1170 CLS
1180 PRINT TAB(5,10);"ANGULO 1";TAB(20);"ANGULO 2"
1190 REPEAT
1200 PROCcalc_angulos
1210 PRINT TAB(5,12);SPC(40)
1220 PRINT TAB(5,12);angulo1;TAB(20);angulo2
1230 PROCdemora(1000)
1240 A$=INKEY$(1)
1250 UNTIL A$<>""
1260 ENDPROC
1280 DEF PROCdefinir_angulos
1290 CLS
1300 PRINT TAB(5,10);:INPUT"Posicion cero 1er. brazo";uno_cero
1310 PRINT TAB(5);:INPUT"Posicion noventa 1er. brazo";uno_noventa
1320 PRINT TAB(5);:INPUT"Posicion cero 2do. brazo";dos_cero
1330 PRINT TAB(5);:INPUT"Posicion noventa 2do. brazo";dos_noventa
1340 uno_factor=90/(uno_noventa-uno_cero)
1350 dos_factor=90/(dos_noventa-dos_cero)
1360 ENDPROC
1380 DEF PROCcalc_angulos
1390 angulo1=INT((ADVAL(1)-uno_cero)*uno_factor)
1400 angulo2=INT((ADVAL(2)-dos_cero)*dos_factor)
1420 DEF PROCdemora(demora)
1430 LOCAL I
1440 FOR I=1 TO demora:NEXT I
1450 ENDPROC
1460 PROCdemora(1000)
```

Kevin Jones

Geometría del brazo



Los desplazamientos horizontales y verticales de la mira del trazador se pueden calcular a partir de los ángulos proporcionados por las lecturas de los dos potenciómetros, utilizando cálculos geométricos simples basados en los dos triángulos rectángulos que podemos apreciar. Las longitudes de x1 e y1 corresponden a la longitud del brazo 1 multiplicadas por COS(angulo 1) y SIN(angulo 1), respectivamente. Dado que el ángulo proporcionado por el segundo potenciómetro está en relación a la posición del brazo 1, las longitudes de x2 e y2 son un poco más difíciles de calcular. El ángulo requerido en el segundo triángulo no es el ángulo 2, sino ángulo 1+ángulo 2, empleándose las funciones COS y SEN junto con la longitud del brazo 2 para calcular los desplazamientos horizontales y verticales. El desplazamiento global de la mira desde el pivote del brazo se puede hallar mediante la suma de x1 y x2, y de y1 e y2

Kevin Jones

zo del programa y utilizarlo en todos los posteriores cálculos de ángulo. Si calculamos:

```
factor=90*/(brazonoventa-brazocero)
```

podemos volver a expresar la fórmula como:

```
angulo=(ADVAL(n)-brazocero)*factor
```

El procedimiento calc_xy calcula los ángulos actuales tomados por los dos potenciómetros y los utiliza para calcular los desplazamientos x e y de la mira. La geometría involucrada es bastante directa. Con respecto al diagrama de geometría del brazo, x1 e y1 se relacionan con la longitud del brazo 1 mediante las funciones SIN y COS:

```
x1=COS(angulo1)*longitud_brazo1
y1=SIN(angulo1)*longitud_brazo1
```

El cálculo de x2 e y2 es un poco más complicado, dado que angulo2 está en cero cuando el brazo 2 se halla en línea con el brazo 1. El ángulo en la esquina del triángulo rectángulo es, por consiguiente, angulo1+angulo2 y:

```
x2=COS(angulo1+angulo2)*longitud_brazo2
y2=SIN(angulo1+angulo2)*longitud_brazo2
```

Los desplazamientos x e y totales se pueden hallar sumando, respectivamente, x1 y x2, e y1 e y2.

Las fórmulas utilizadas en el procedimiento cal_xy son ligeramente diferentes de las reseñadas anteriormente. Las variables de longitud del brazo se reemplazan por variables escaladas. Estas variables están relacionadas con las longitudes de los dos brazos, pero también incorporan un factor de modo que los desplazamientos x e y se realicen a escala para que quepan en el sistema de coordenadas de gráficos del BBC Micro.

La forma más obvia de guardar visualizaciones creadas en la pantalla es realizar en código máquina una salvaguardia del área de RAM utilizada por MODE 1. Siempre y cuando no se haya desplazado la pantalla desde el último CLS, esta zona estará entre &3000 y &7FFF. Sin embargo, el empleo de la instrucción *SAVE tiene el inconveniente de que no se le pueden pasar nombres de archivo utilizando una variable en serie. Por ejemplo, la instrucción:

```
*SAVE archivo$ 3000 8000
```

guardará el archivo con el nombre archivo$ en vez del nombre en serie retenido en la variable archivo$. Esto significa que el usuario no puede guardar fácilmente visualizaciones en pantalla en disco o cinta bajo diferentes nombres de archivo. Por fortuna, existe una forma de evitar este problema, empleando la llamada al sistema operativo OSCLI. Esta instrucción se puede utilizar para ejecutar un bloque de códigos ASCII retenido en la memoria, como si los caracteres correspondientes se hubieran entrado directamente desde el teclado. Para usar esta llamada debemos pasar en los registros X e Y las direcciones de comienzo del bloque ASCII en forma *lo-byte/hi-byte*. Para llevar a cabo *SAVE y *LOAD, por lo tanto, ensamblamos la instrucción (junto con el nombre de archivo proporcionado por el usuario) en una serie y colocamos (POKE) los valores ASCII de los caracteres que componen la serie en un bloque reservado de memoria. Tras establecer los registros X e Y de modo que apunten al comienzo de este bloque, la instrucción allí almacenada se lleva a cabo llamando a OSCLI en la dirección &FFF7. Esta es la función que cumple el procedimiento instruccion_oscli de la línea 2170.

Además de guardar y cargar pantallas, la pantalla se puede limpiar digitando C, y regresar al menú principal digitando M. La segunda selección de este menú no se puede efectuar en esta etapa, puesto que constituirá la segunda parte del programa y la ofreceremos en el próximo capítulo. Se pueden seleccionar nuevos colores de primer plano en cualquier momento pulsando 1, 2 o 3. Estos colores son

los de primer plano por defecto de MODE 1, y se los puede cambiar mediante el empleo de la instrucción VDU 19. El programa que ofrecemos se ejecutará correctamente en modalidad de mano alzada.

Programa del trazador

La primera mitad del programa *Trazador digital*, que ofrecemos aquí, permite utilizar el trazador a mano alzada. La segunda mitad incluye rutinas que permitirán emplearlo para producir puntos, líneas y curvas individuales. En modalidad de mano alzada, hay seis facilidades disponibles. El botón a presión, montado en el trazador, actúa como un mando de "lápiz arriba/lápiz abajo", permitiendo que el usuario pase de uno a otro. El botón está cableado en la puerta analógica como el botón de disparo de una palanca de mando y se puede detectar desde software examinando los dos bits inferiores del valor devuelto en ADVAL(0). Cada uno de los dos bits estará establecido en uno si se pulsa el botón de disparo correspondiente. Por tanto, (ADVAL(0)AND3) <> 0 indica que se ha producido la pulsación de un botón de disparo y se puede emprender la acción adecuada. En este caso, una variable denominada flagart oscilará entre 0 y 1 a cada sucesiva pulsación del botón. El valor de flagart se comprueba en el procedimiento dibujar y se toma la decisión ya sea de DRAW o bien de MOVE a una nueva posición (según cual sea el valor).

Se utiliza una pequeña cruz como cursor para indicar la posición actual de la mira del trazador en la pantalla. La misma se puede borrar y desplazar sin alterar los datos de fondo mediante el uso de la modalidad de trazado Exclusive-OR. Todas las líneas dibujadas en esta modalidad (seleccionada mediante GCOL3) se pueden borrar volviendo a dibujarlas exactamente en la misma posición. Por lo tanto, el procedimiento dibujar calcula las nuevas coordenadas del curso y después llama a un procedimiento para dibujar el cursor, que borrará el cursor viejo volviéndolo a dibujar en modalidad Exclusive-OR. El cursor de gráficos (invisible para nosotros) se desplaza (MOVE) hasta la antigua posición del cursor y se dibuja una línea o se realiza un desplazamiento hasta las nuevas coordenadas, según cual sea el valor de flagart. Por último, los valores actuales de las coordenadas se almacenan en antx y anty, como preparación para la próxima ocasión en que se llame al procedimiento dibujar.

```
   6 REM ** TRAZADOR DIGITAL BBC **
1080 PROCdefinir_parametros
1090 REPEAT
1100 MODE 1
1110 REM APAGAR CURSOR
1120 VDU 23,1,0;0;0;0;
1130 PROCmenu
1140 CLS
1150 IF resp$="1" THEN PROCmanoalzada ELSE PROCelastico
1160 UNTIL flagfinal=1
1170 END
```

Se proporcionan las longs. de los brazos y las lecturas de calibrado de los pots.

```
1190 DEF PROCdefinir_parametros
1200 uno_cero=14820
1210 uno_noventa=45020
1220 uno_longitud=250
1230 uno_factor=90/(uno_noventa-uno_cero)
1240 dos_cero=450
1250 dos_noventa=25300
1260 dos_longitud=222+28
1270 dos_factor=90/(dos_noventa-dos_cero)
1280 REM ** FACTORES DE CONVERSION DE MM A COORDENADAS
     GRAFICOS **
1290 escala=1023/460
1300 uno_escala=escala* uno_longitud
1310 dos_escala=escala* dos_longitud
1320 REM EXPLORAR SOLO 2 CANALES ADC
1330 *FX 16,2
1340 flagfinal=0:flagart=0:color=1
1350 DIM save% 30:DIM x(3),y(3)
1360 ENDPROC
1380 DEF PROCmanoalzada
1390 PROC free_inform
1400 REPEAT
1410 IF(ADVAL(0) DIV 256)<>0 THEN PROCdibujar
1420 IF(ADVAL(0)AND 3)<>0 THEN PROCarticular_lapiz
1430 resp$=INKEY$(1):IF resp$<> "" THEN PROCpulsacionlibre
1440 UNTIL flagsalida=1
1450 ENDPROC
1470 DEF PROCdibujar
1480 PROCcalc_xy
1490 PROCcursor(antx,anty)
1500 MOVE antx,anty
1510 IF flagart=0 THEN DRAW x,y ELSE MOVE x,y
1520 PROCcursor(x,y)
1530 antx=x:anty=y
1540 ENDPROC
1560 DEF PROCcursor(cx,cy)
1570 GCOL 3,3:REM MODALIDAD TRAZADO EOR
1580 MOVE cx,cy-16
1590 PLOT 1,0,32
1600 PLOT 0,-16,-16
1610 PLOT 1,32,0
1620 GCOL 0,color
1630 ENDPROC
```

El sig. proc. lleva a cabo los cálculos para convertir las lecturas de los pots. en comps. verticales y horizontales x e y

```
1650 DEF PROCcalc_xy
1660 angulo1=RAD((ADVAL(1)-uno_cero)*uno_factor)
1670 angulo2=RAD((ADVAL(2)-dos_cero)*dos_factor)
1680 x=uno_escala*COS(angulo1)+dos_
     escala*COS(angulo1+angulo2)
1690 y=1023-(uno_escala*SIN(angulo1)+dos_
     escala*SIN(angulo1+angulo2))
1700 ENDPROC
1720 DEF PROCmenu
1730 flagsalida=0:flagfinal=0
1740 PRINT TAB(5,10);"Por favor, elija"
1750 PRINT TAB(5);"1...Modalidad mano alzada"
1760 PRINT TAB(5);"2...Modalidad elastica"
1770 PRINT:PRINT TAB(5);"Pulse 1 o 2"
1780 REPEAT:resp$=GET$:UNTIL resp$="1" OR resp$="2"
1790 ENDPROC
1810 DEF PROCarticular_lapiz
1820 flagart=1-flagart
1830 REPEAT UNTIL (ADVAL(0) AND 3)=0
1840 ENDPROC
1860 DEF PROCpulsacionlibre
1870 IF resp$="C" THEN CLS:PROCfree_inform:ENDPROC
1880 IF resp$="M" THEN flagsalida=1
1890 IF resp$"S" THEN PROCguardar_pantalla:PROCfree_
     inform:ENDPROC
1900 IF resp$="L" THEN PROCcargar_pantalla:PROCfree_
     inform:ENDPROC
1910 PROCcambio_color
1920 ENDPROC
1940 DEF PROCfree_inform
1950 PROCcal_xy:antx=x:anty=y:PROCcursor(antx,anty)
1960 PRINT TAB(1,1);SPC(79)
1970 PRINT TAB(1,1);"S=Save L=Load M=Menu C=Limpiar"
1980 GCOL 0,1:MOVE 0,920:DRAW 1280,920
1990 ENDPROC
2010 DEF PROCcambio_color
2020 resp=VAL(resp$)
2030 IF resp<1 OR resp>3 THEN ENDPROC
2040 color=resp
2050 ENDPROC
```

Estos procs. permiten guardar o cargar el área de pant. como área RAM usando *SAVE y *LOAD indirectamente a través de OSCLI

```
2070 DEF PROCguardar_pantalla
2080 PROCcursor(x,y):REM CURSOR APAGADO
2090 REPEAT
2100 PRINT TAB (1,1);SPC(79)
2110 INPUT TAB (1,1);"GUARDAR
     NOMBREARCHIVO";archivo$:archivo$=archivo$+".S"
2120 UNTIL LEN(archivo$)<8
2130 archivo$="*SAVE"+archivo$+"3000 8000"
2140 PROCinstruccion_oscli(archivo$)
2150 ENDPROC
2170 DEF PROCinstruccion_oscli(a$)
2180 FOR I=0 TO LEN(a$)-1
2190 save%?I=ASC(MID$(a$,I+1,1))
2200 NEXT I
2210 save%?I=13:REM AÑADIR CR
2220 X%save% MOD 256:Y%=save% DIV 256
2230 CALL &FFF7:REM LLAMAR OSCLI
2240 ENDPROC
2260 DEF PROCcargar_pantalla
2270 REPEAT
2280 PRINT TAB(1,1);SPC(79)
2290 INPUT TAB(1,1);"LOAD
     NOMBREARCHIVO";archivo$:archivo$=archivo$+".S"
2300 UNTIL LEN(archivo$)<8
2310 archivo$="*LOAD"+archivo$
2320 PROCinstruccion_oscli(archivo$)
2330 ENDPROC
```

En la Luna

Este juego ha sido escrito por Paul Dunning para el microordenador Atari. Para posar su nave espacial en la superficie lunar es preciso utilizar la palanca de mando.

Su misión consiste en posar su nave espacial en una de las tres pistas azules que se visualizan en la pantalla. Si la velocidad es excesiva usted se estrellará; una velocidad de 5 parece adecuada. Su puntuación se calcula en función de la velocidad a la cual aluniza y de la pista que haya escogido. Si tiene éxito en su alunizaje, vuelve a partir con un suplemento de 500 litros de combustible. El juego finaliza cuando haya agotado su stock de carburante.

```
0 REM * ALUNIZAJE DE PAUL DUNNING *
1 DATA 108, 60, 81, 40, 53, 40, 60, 40, 64, 40, 72, 40, 40, 40, 53,
  40, 60, 40, 64, 40, 72, 40, 40, 40, 53, 40, 60, 40, 64, 40, 60, 40,
  72, 60
2 DATA 108, 60, 96, 80, 60, 40, 64, 40, 72, 40, 81, 80, 72, 40, 64,
  40, 72, 60, 96, 40, 85, 40, 53, 80, 40, 40, 45, 40, 50, 40, 53, 40,
  60, 40
3 DATA 68, 40, 72, 40, 81, 40, 53, 60, -1
4 CLR :DIM AS(100), HS(11), E$(60):HS = " a Lari":POKE 752,
  1:RE=500
5 AA=1:CA=4000: TE=100:SC=0:AN=0:VI=AN:POKE 53278,
  1:GOSUB 900:GOSUB 4000
6 DATA 11, 12, 13, 13, 14, 15, 17, 19, 21, 22, 23, 23, 24
7 DATA 25, 26, 29, 30, 31, 33, 33, 33, 33, 33, 33, 31
8 DATA 31, 30, 29, 28, 26, 25, 25, 24, 23, 21, 16, 16, 15
9 DATA 14, 14, 14, 15, 17, 19, 21, 23, 24, 25, 26, 27, 29
10 DATA 29, 29, 29, 29, 27, 26, 26, 25, 26, 27, 28, 28
11 DATA 27, 26, 25, 25, 23, 23, 23, 23, 23, 23, 23, 21, 21
12 DATA 20, 18, 17, 17
15 FOR I=1 TO 8:POKE 53247+I,0:NEXT I:GOTO 495
30 END
100 REM POKE 53248, 30
200 REM ***** JUEGO JUGADOR *****
210 POKE 559,0:POKE 710, 148:POKE 712,0
215 POKE 752,1:AL=110
220 RESTORE 235:FOR S2=1 TO 60:READ S3:E$(S2,S2)=CHR$
    (S3):NEXT S2
235 DATA 104, 104, 104, 74, 72, 176, 11, 160, 2, 177, 203, 136,
    145, 203, 200, 200, 208, 247, 104, 74, 72
236 DATA 176, 11, 160, 253, 177, 203, 200, 145, 203, 136
237 DATA 136, 208, 247, 104, 74, 72, 176, 7, 198, 205
238 DATA 165, 205, 141, 0, 208, 104, 74, 176, 7, 230, 205, 165,
    205, 141, 0, 208, 96, 0, 0
240 A=PEEK(106)-16:POKE  54279,A:POKE204,  A+4:POKE  203,
    0:PMBASE=A*256
250 POKE 53277, 3:POKE 704,40:POKE 53248, 150:POKE 205,
    0:POKE 53256, 0
260 FOR I=PMBASE-1024 TO PMBASE+1280:POKE I, 0:NEXT I
270 RESTORE 340
280 FOR I=PMBASE+1152 TO PMBASE+1209:READ B:POKE I,
    B:NEXT I
340 DATA 0, 60, 60, 126, 126, 195, 195, 126, 126, 60, 60, 24, 24,
    126, 126, 195, 195
350 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
360 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
370 POKE 559,46
380 IF STRIG(0)=0 THEN GOSUB 500
390 POKE 559,0
395 IF AA=1 THEN FOR I=4 TO 1014:A=USR(ADR(E$), 14):SOUND
    0, I, 10,4:NEXT I:AA=2:A=USR(ADR(E$), 11):SOUND
    0,0,0,0
400 POKE 559,46
410 IF CA<=AN AND CA>=30 THEN SOUND 0,255, 8, 15:GOTO 420
415 SOUND 0,0,0,0
420 IF CA<=AN+30 AND CA>=AN+5 THEN SOUND 1,100,8,3:GOTO
    430
425 SOUND 1,0,0,0
430 POKE 53278,0:IF CA<=AN THEN GOTO 455
431 IF ST=7 THEN AN=AN+0.05:CA=CA-5
432 IF ST=11 THEN AN=AN-0.05:CA=CA-5
440 HP=HP+AN: IF HP>1 THEN HP=0:SR=7
450 IF HP<-1 THEN HP=0:SR=11
455 VI=VI+0.02:VP=VP+VI:IF  VP>1  AND  SR=1  5  THEN
    VP=0:SR=13:AL=AL-1
456 IF VP>1 AND SR=11 THEN VP=0:SR=9:AL=AL-1
457 IF VP>1 AND SR=7 THEN VP=0:SR=5:AL=A-L-1
458 IF VP<-1 THEN VP=0:SR=14:AL=AL+1
460 IF ST=14 AND CA>29 THEN VI=VI-0.05:CA=CA-30
480 A=USR(ADR(E$),SR):SR=15
485 ? " PUNTUACION=";SC; ";CHR$(127);" VEL.=" ;VI*100;" "
486 ? " CARB =";CA;" ";CHR$(127);" ANG =";AN*100;"
487 ? "TIEMPO =";INT(TE);" ";CHR$(127);"ALT =";AL;" "
488 ? CHR$(28);CHR$(28);CHR$(28);
489 TE=TE-0,1
490 IF TE=0 THEN 720
492 GOTO 370
495 GOTO 210
500 IF CO=4 AND VI>0.5 THEN 600
505 IF CO=4 AND VI>0.4 THEN 700
510 IF CO=2 THEN 700
520 RETURN
600 FOR Q=1 TO 2:FOR W=255 TO 100 STEP -1
610 SOUND 0,W,10,15:SOUND 1,W-10,10,15:SOUND 2,W-20,10,15
620 NEXT W:NEXT Q
630 SS=0.5-VI:SO=SC:IF SS=0 THEN SC=SC+5 0:GOTO 650
640 IF AL<5 THEN SC=SC+(SS*100)*10:GOTO 650
642 IF AL<20 THEN SC=SC+(SS*100)*20:GOTO 650
644 IF AL<60 THEN SC=SC+(SS*100)*5
650 IF CA<2500 THEN CA=CA+(SC-SO*2)
655 FOR Q=0 TO 3:SOUND Q,0,0,0:NEXT Q:AA=1:POKE 53278,
    1:GOSUB 800:GOTO 210
700 FOR Q=0 TO 50:POKE 712, RND(1)*255:SOUND 0,255, 8,
    15:SOUND 1,240,8,15:SOUND 2,220,8,15:NEXT 0
705 SOUND 0,0,0,0:SOUND 1,0,0,0:SOUND 2,0,0,0
706 IF CA<30 THEN CA=0:GOTO 720
710 POKE 712,0:AA=1:GOSUB 800:GOTO 210
720 POKE 712,0:GOTO 810
800 VP=0:HP=0:AN=0:VI=0:TE=100:RETURN
810 IF SC>RE THEN GOSUB 1010
820 GRAPHICS 17:SETCOLOR 4,1,0
830 ? #6;"  FINALIZADO"
835 ? #6:? #6;"       RECORD ";RE
836 ? #6:? #6;"       ";HS
840 ? #6:? #6;"       PULSE START "
850 POKE 53248,0
860 IF PEEK(53279)=6 THEN RESTORE :GOTO 5
870 GOTO 860
899 END
900 REM INTRO
910 GRAPHICS 17:SETCOLOR 4,1,0:E=0
920 ? #6;"       ALUNIZAJE "
930 ? #6:? #6;" DE P. DUNNING SEP.82"
940 ? #6:? #6;"PALANCA DE MANDO (0)"
950 ? #6:? #6;"         ALTO          "
960 ? #6:? #6;"
970 ? #6;"  IZQUIERDA<      >DERECHA"
980 ? #6
985 POKE 53248,0
990 READ M:IF M<0 THEN 998
992 READ P
995 SOUND 0,M,10,15:FOR PP=0 TO P:NEXT PP
997 GOTO 990
998 ? #6;"       buena suerte"
999 FOR  X=1  TO  13:FOR  Y=0  TO  255  STEP  20:SOUND
    0,Y,10,15:NEXT Y:NEXT X:SOUND 0,0,0,0:RETURN
1000 RESTORE 1:GOTO 990
1010 HS=" ":TE=30:GRAPHICS 18:SETCOLOR 4,1,0:? #6;"     NO
     ESTA MAL  ":? #6:X=1:Y=1:RE=SC
1011 ? #6;"  a b c d e f g h "
1012 ? #6
1013 ? #6;"  j j k l m n o p "
1014 ? #6
1015 ? #6;"  q r s t u v w x "
1016 ? #6
1017 ? #6;"  y z . -      R E "
1018 POKE 53248,0
1020 FOR     Q=10     TO     245:SOUND     0,Q-10,8,15:SOUND
     1,Q,8,15:SOUND 2,Q+10,8,15:NEXT Q
1030 POSITION X,Y:? #6;"XXX": POSITION X,Y+1:? #6;"X": POSI-
     TION X+2,Y+1:? #6;"X": POSITION X,Y+2:? #6;"XXX"
1040 IF STICK(0)<>7 THEN 1050
1045 POSITION X,Y:? #6;"     ": POSITION X,Y+1:? #6;"  ": PO-
     SITION X+2,Y+1:? #6;" ":POSITION X,Y+2:? #6;"  "
1046 IF X>13 AND Y=7 THEN X=1:Y=1:GOTO 1050
1047 IF X>13 THEN X=1:Y=Y+2:GOTO 1050
1048 X=X+2
1050 IF STICK(0)<> 11 THEN 1060
1055 POSITION X,Y:? #6;"":POSITION X,Y+1:? #6;"":POSITION
     X+2,Y+1:? #6;"":POSITION X,Y+2:? #6;"  "
1056 IF X<3 AND Y=1 THEN X=15:Y=7:GOTO 1060
1057 IF X<3 THEN X=15:Y=Y-2:GOTO 1060
1058 X=X-2
1060 POSITION 6,10:? #6; "TIEMPO ";INT(TE);" ":TE=TE-0.1:IF
     TE=0 THEN 1100
1065 FOR PP=0 TO 20:NEXT PP
1069 IF STRIG(0)<>0 THEN 1080
1071 LOCATE X+1,Y+1, XX:IF XX = ASC("E") THEN 1100
1072 IF   XX=ASC("R")   AND   LEN(H$)>1   THEN   H$   =
     H$(1,LEN(H$)-1>:GOTO 1080
1073 IF LEN(H$)>10 THEN 1080
1075 H$(LEN(H$)+1)=CHR$(XX)
1080 POSITION 4,11:? #6,H$;" "
1099 GOTO 1030
1100 FOR Q=0 TO 3:SOUND Q,0,0,0:NEXT Q
1110 RETURN
4000 GRAPHICS 5
4010 RESTORE 6:COLOR 2
4020 FOR X=0 TO 79
4030 READ Y
4040 PLOT X,39:DRAWTO X,Y
4050 NEXT X
4060 COLOR 3
4070 PLOT 18,32:DRAWTO 23,32
4080 PLOT 50,28:DRAWTO 54,28
4090 PLOT 67,22:DRAWTO 73,22
4100 RETURN
```

anana sensible
Banana Interface, así llamada
r los enchufes banana que
liza, permite la conexión en
erface de dispositivos
ctricos analógicos con un
enador (que utiliza señales
tales). De esta forma se
den construir sistemas de
mentación que permitan que
denador controle
tecimientos del mundo
ior y responda a ellos

Bananarama

Bajo su piel
La Banana Interface se puede utilizar para una gran variedad de aplicaciones de control, tanto en el hogar como en el laboratorio. Por ejemplo, el entorno de la habitación que vemos en la ilustración se puede poner casi por completo bajo el control de un ordenador conectado a una Banana Interface. En el cuarto se pueden instalar dispositivos de percepción, tales como termostatos o sensores luminosos. La información recibida desde estos dispositivos las puede analizar el ordenador y compararla con rangos óptimos preprogramados. Cualquier ajuste del entorno que resulte necesario (como elevar la temperatura, bajar las persianas o encender las luces por la tarde) se puede llevar a cabo a través de las líneas de salida de la interface

SC Sensor de contacto
CC Control de calefacción
IL Interruptor para reducir la luz
ME Motor eléctrico
AA Altavoz de alarma
DH Detector de humo
TM Termostato
CF Célula fotosensible

La Banana Interface constituye un interesante aporte en el campo de los dispositivos controlados por ordenador

Anteriormente ya hemos examinado numerosos dispositivos que permiten que los ordenadores controlen maquinarias, y en nuestro apartado *Bricolaje* también hemos ilustrado cómo se pueden controlar numerosos dispositivos. Uno de los últimos que han salido al mercado, dirigido básicamente a las escuelas, es la Banana Interface, diseñada para permitir controlar varios proyectos diferentes desde el BBC Micro o el Commodore 64.

Esencialmente el concepto que subyace en la Banana Interface es idéntico al de la caja buffer que construimos en el apartado *Bricolaje*, si bien el Banana es algo más sofisticado. Manipulando los patrones de bits binarios que contienen los registros de datos y de dirección de datos del ordenador, se pueden transmitir señales de 5 V a dispositivos externos. Además, debido a que las puertas para el usuario y la impresora son bidireccionales, la máquina puede recibir señales desde el periférico que le permiten "saber" algo sobre el estado del dispositivo (su posición, p. ej.) o si se ha activado algún interruptor disparador. Por consiguiente, es posible que el ordenador pueda, mediante una combinación de señales de entrada y salida, guiar con precisión a un robot y reaccionar ante las circunstancias cambiantes del entorno del mismo.

La interface se conecta al ordenador a través de un trozo de cable plano que se enchufa en las puertas para la impresora y para el usuario. El otro extremo del cable está instalado en un conector de 40 vías de la propia interface. Para poder operar la Banana Interface se requieren dos fuentes de alimentación eléctrica CD, la primera de las cuales es una corriente de 5 V que se suministra desde el ordenador a través del cable plano. Esta corriente se utiliza para ejecutar la lógica de la interface a lo largo de las líneas de datos.

La segunda fuente de alimentación se proporciona externamente mediante un par de conectores *minijack* situados en la parte posterior del dispositivo junto a la interface del cable plano. Lamentablemente, el fabricante, Castle Associates, ha decidido no proporcionar con la interface una fuente de 12 V. La razón de ello es que el Banana está destinado básicamente para ser usado en las escuelas, las cuales suelen contar con amplios suministros de

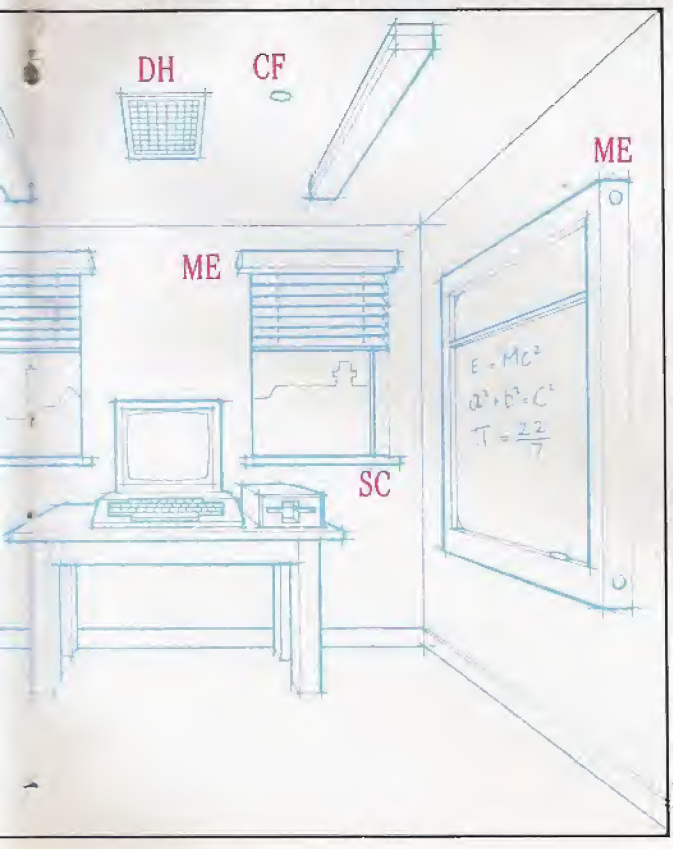
la corriente adecuada en sus laboratorios y talleres. Por consiguiente, dado que las escuelas no necesitan una fuente de alimentación externa, la empresa ha optado por excluirla y mantener reducido el precio del dispositivo.

Esto significa que los particulares interesados en adquirir un Banana se encuentran ante la perspectiva de tener que conseguir su propia fuente de alimentación. Puesto que las fuentes de 12 V con pequeños amperajes no se consiguen con facilidad, probablemente se tendrá que operar la interface mediante unas cuantas pilas conectadas en serie (la empresa recomienda un amperaje de 600 mA, aunque la máquina que utilizamos operó sin dificultades con una fuente de 10 V y un amperio).

Cuando la Banana Interface se hallaba en fase de planificación, Castle Associates consultó a profesores de diseño y tecnología. Esta consulta repercutió positivamente en las especificaciones del dispositivo, en particular en cuanto concierne a su uso en las escuelas. Con la carcasa completamente construida en acero, la interface es uno de los dispositivos más robustos que existen para cualquier ordenador personal.

Esta preocupación por la duración también se ha aplicado a los componentes internos de la interface. La placa de circuito impreso se ha soldado a los conectores hembra, manteniendo firmemente en su sitio el delicado tablero. El resultado es un periférico que probablemente pueda soportar sin problemas las manipulaciones erróneas y que, para ser objeto de un daño serio, requerirá la intervención de un escolar muy decidido a lograrlo.

En la parte superior de la carcasa hay varias hileras de conectores *minijack* divididos en tres grupos. A la izquierda de la interface hay ocho líneas de entrada blancas, con cuatro líneas a tierra, verdes, colocadas entremedio. Cada enchufe hembra está numerado 1, 2, 4, 8, y así sucesivamente, que corresponden al número retenido en cada una de las posiciones de bits del registro. Junto a las posiciones de bits hay LEDs que se utilizan para dar información respecto a cuál de los bits está *high* (indicando que a través de esa línea de datos concreta está pasando una corriente de 5 V).



Kevin Jones

El ordenador puede registrar los cambios en las líneas de entrada, y la forma de demostrarlo es estableciendo *high* todas las líneas. Hacer un PEEK del registro producirá un valor de 255, pero si conectáramos un cable desde una de las líneas de tierra a uno de los conectores de entrada blancos, el voltaje caería a cero y el valor del registro caería consecuentemente, según cuál de las líneas se utilizara. Por lo tanto, se podrían construir aplicaciones simples de interrupción de circuitos, como alarmas antirrobo.

Las líneas de salida

A la derecha de las entradas, y ocupando la mayor parte de la superficie, están las líneas de salida. Al igual que las de entrada, hay ocho posiciones básicas (numeradas 1, 2, 4, y así hasta 128), LEDs correspondientes a cada una de las posiciones de bits y cuatro conectores *minijack*. Mediante circuitos de relé se conectan pares de conectores hembra. La colocación (POKE) de un número en el registro en la dirección &FE61 disparará los relés.

Conectando los motores eléctricos y fuentes de alimentación adecuadas a través de los terminales, estos relés se pueden utilizar para activar los motores y posicionarlos como se desee. Empleándola de esta forma, desde la Banana Interface se pueden controlar hasta cuatro motores eléctricos a la vez. Obviamente, la forma más eficaz de controlar los motores eléctricos (que se podrían conectar a una tortuga para el suelo o a un brazo-robot) es activarlos a través de los interruptores de salida y monitorizar sus movimientos a través de las puertas de entrada.

A lo largo de la parte superior de la interface hay otro grupo de ocho puertas lógicas de alta velocidad, cada una de las cuales produce una corriente de 12 V. Estas líneas de salida se utilizan para activar desde la interface motores paso a paso pudiéndose activar hasta siete de ellos al mismo tiempo.

Parece que la Banana Interface puede tener un buen mercado en las escuelas y los centros de investigación como medio auxiliar para la enseñanza del control básico por ordenador de dispositivos eléctricos. Es claro que existen numerosas aplicaciones para las cuales se podría utilizar en demostraciones y experimentos. También podría tener amplias aplicaciones en pequeñas empresas que deseen automatizar algunos de sus procesos de producción.

Desde el punto de vista del aficionado, la utilidad de un dispositivo de estas características ya no está tan clara. Ciertamente se lo puede emplear en casa para numerosos experimentos realizados como pasatiempo y es probable que le enseñe muchísimas cosas sobre las técnicas de control por ordenador. En este sentido, el amante serio de la robótica bien puede considerar la adquisición de uno. No obstante, a los usuarios del BBC Micro y del Commodore 64 cuyo interés por estos dispositivos sea sólo superficial, la Banana Interface les resultará excesivamente cara por sus limitaciones.

BANANA INTERFACE

DIMENSIONES

300×200×63 mm

INTERFACES

Conector en paralelo de 40 vías desde las puertas para la impresora y el usuario del BBC Micro y el Commodore 64. Puertas de salida, ocho conectores de salida de 12 V

VENTAJAS

La Banana Interface está bien construida y permite una amplia variedad de aplicaciones de control por ordenador

DESVENTAJAS

La falta de una fuente de alimentación de 12 V adecuada limita su mercado potencial ya sea a los aficionados serios o bien a los establecimientos educativos o de investigación

Interés declarado

Examinaremos la naturaleza declarativa del PROLOG y la forma como maneja proposiciones.

El PROLOG está diseñado para dar al programador un medio de describir la estructura lógica de un problema utilizando una base de datos de hechos y reglas *if-then*. Esta enunciación "declarativa" del problema la emplean los mecanismos deductivos del PROLOG para producir respuestas a los interrogantes que uno ha planteado. La tarea del programador se transforma, por consiguiente, de tener que decirle al ordenador, con todo detalle, cómo usar las operaciones de un lenguaje (para resolver un conjunto predefinido de problemas) a tener que proporcionar una enunciación clara y lógica del conocimiento requerido para resolver los problemas, dejando luego que sea el PROLOG el que lleve a cabo la tarea.

La sintaxis del PROLOG es muy sencilla, pero la terminología es bastante oscura. La construcción más importante es el *término*. Un término puede ser una *constante* (como "árbol", "juan" o "25"), una *variable* o una *estructura*. Las estructuras se crean a partir de constantes y variables. Una estructura muy común en los programas en PROLOG es el *hecho*. Éste se compone de un predicado (que debe ser una constante) ya sea solo, o bien seguido de una lista de argumentos (que pueden ser constantes o variables) entre paréntesis:

```
predicado1.
predicado2(argumento1,argumento2,
argumento3).
```

Como vimos en el primer capítulo de esta serie, se puede considerar al predicado como una relación existente entre sus argumentos. En la frase en castellano "Venusianos comen árboles", "comen" es el predicado que relaciona a venusianos con árboles. En PROLOG, podríamos escribir este hecho como:

```
come(venusianos,árboles).
```

Los términos como éste se pueden acumular para formar grandes grupos de hechos de forma muy similar a como se acumulan registros en archivos de bases de datos. En el diagrama tenemos una base de datos de hechos acerca de quién come qué. Sin ningún código extra, esta base de datos ya nos ofrece un programa en PROLOG.

Para ejecutar el programa necesitamos establecer una proposición para que el intérprete la demuestre.

¿Quién come qué?

```
come(venusianos,árboles).
come(jovianos,rocas).
come(terrícolas,gachas).
come(mercurianos,gusanos).
come(neptunianos,gachas).
come(marcianos,marcianos).
```

En PROLOG, los "hechos" se expresan como un predicado (en este caso, "come"), en solitario o seguido por una lista de argumentos. Se los puede utilizar para proporcionar al programa una "base de datos de conocimientos" a la cual pueda interrogar luego el usuario

Supongamos que deseamos saber si los jovianos comen rocas. Planteamos el interrogante digitando:

```
?-come(jovianos,rocas).
```

tras el aviso del sistema (por lo general ?-). El PROLOG examina entonces nuestros hechos y, si encuentra una pareja para la proposición, dice yes, lo que significa "sí, a partir del programa se puede ver que come(jovianos,rocas) es verdadero". Si después preguntamos:

```
?-come(marcianos,gachas).
```

el PROLOG responderá no.

Podemos lograr que el PROLOG nos haga más cosas empleando variables en nuestra interrogación. La convención del PROLOG es que los nombres de variables empiezan con una letra en mayúscula, mientras que las constantes estándares empiezan con minúscula. Por lo tanto, si deseamos saber quiénes comen gusanos, podemos preguntar:

```
?-come(Criatura,gusanos).
```

a lo que el PROLOG responderá:

```
Criatura=mercurianos
```

y después hará una pausa, esperando una entrada. Ello significa que el PROLOG ha descubierto que, estableciendo la variable Criatura en el valor mercurianos, puede probar que la proposición es verdadera.

En este punto podemos ya sea entrar RETURN, para indicar que estamos satisfechos con la respuesta, o bien podemos digitar un punto y coma para indicarle al PROLOG que busque otra forma de demostrar la proposición. En este ejemplo, sin embargo, no hay ninguna otra forma, de modo que el PROLOG nos responderá no. No obstante, si le solicitamos que busque quién come gachas, habrá dos respuestas posibles: terrícolas y neptunianos. De modo que, en respuesta a:

```
?-come(Criatura,gachas).
```

el PROLOG dice:

```
Criatura=terrícolas
```

y después digitamos ;, a lo que el PROLOG responderá:

```
Criatura=neptunianos
```

Si hubiera más soluciones, la pulsación del punto y coma tras cada una obligaría al PROLOG a hallarlas todas. En el próximo capítulo veremos exactamente cómo funciona esto, pero veamos antes otros tres importantes conceptos: *reglas, cadenas de inferencia* y *retroceso*.

Uno o más términos constituyen una cláusula, que puede expresar una *regla*. Una cláusula siempre posee un encabezamiento compuesto por un término, y éste puede ir seguido por su cuerpo, formado por uno o más términos. El encabezamiento y el cuerpo se separan mediante el operador simbólico :-, que normalmente se lee como "si". Por consiguiente, tenemos:

```
término1:-término2,término3,término4.
```

Las comas entre los términos del cuerpo se pueden leer como un AND lógico, de modo que se podría interpretar que esta estructura de nuestro ejemplo significa que "el término1 es válido SI el término2 AND el término3 AND el término4 son todos válidos".

Para clarificar mejor esto, imaginemos que deseamos averiguar si alguna de las criaturas de nuestra base de datos es caníbal. Podemos escribir una cláusula que nos dé una definición de caníbal:

```
caníbal(Criatura):-
come(Criatura,Criatura).
```

Ésta es una cláusula con un término en su cuerpo y podemos leerla como una "regla" que afirma que: una criatura es caníbal si esa criatura come el mismo tipo de criatura. Si añadimos esta regla y después preguntamos:

```
?-caníbal(X).
```

el PROLOG empareja esta proposición con el encabezamiento de nuestra nueva cláusula. Dado que esta nueva cláusula

no es verdadera de forma automática, primero se debe demostrar que los términos del cuerpo de la cláusula son verdaderos. De modo que el PROLOG toma uno por uno, de izquierda a derecha, y los establece como proposiciones, tal como si se los hubiera digitado en forma de interrogaciones. En el cuerpo hay un solo término:

```
...,come(Criatura,Criatura).
```

y concuerda con el hecho (come(marcianos,marcianos). Esto a su vez establece a X=marcianos (X era el nombre de variable de la interrogación original) y el PROLOG responderá:

```
X=marcianos
```

En este caso, come(marcianos,marcianos) se encontró como un hecho en la base de datos. Sin embargo, si no hubiese sido un simple hecho sino otra regla, el PROLOG habría tenido que abocarse a otras subproposiciones para tratar de demostrar la veracidad del encabezamiento de la regla.

El programa en PROLOG del diagrama ilustra cómo podría suceder esto. Si planteamos el interrogante:

```
?-color_de(marty,Color).
```

queriendo averiguar el color de marty, el PROLOG primero toma nuestra interrogación como su proposición.

Luego encuentra una regla para decidir si color_de(marty,rosa) es verdadero y encuentra que ello es así si (:-) el talante de marty es feliz. Estableciendo luego esto como su siguiente proposición, descubre una regla que afirma que marty es feliz si puede programar en PROLOG. De modo que

¿Cuándo se pone azul un marciano?

```
color_de(Marciano,rosa):-talante_de
  (Marciano,feliz)
color_de(Marciano,azul)
talante_de(Marciano,feliz):-
puede_programar_en(Marciano,prolog).
talante_de(Marciano,triste).

puede_programar_en(marty,basic).
puede_programar_en(miranda,prolog)
```

En este conjunto de términos, observe que los nombres de las variables empiezan con mayúscula. El símbolo :- se puede interpretar como un *if* (si) lógico.

puede_programar_en(marty,prolog) se convierte en la siguiente proposición.

Esto podría continuar de este modo a través de muchísimos niveles; pero, en este caso, se interrumpe aquí porque no hay ninguna regla ni ningún hecho que demuestre, o bien que indique cómo demostrar, que marty es un programador de PROLOG.

El PROLOG, simplemente, no se da por vencido en este punto. Lo que hace, sin embargo, es admitir su fracaso en la proposición actual y luego retornar a la proposición previa para ver si hay alguna forma alternativa de demostrarlo. Pero eso también falla (no se puede demostrar que el talante de marty sea feliz). De modo que el PROLOG *retrocede* nuevamente para ver si hay alguna forma alternativa de demostrar la primera proposición. De hecho, sí la hay. El PROLOG encuentra el hecho color_de(marciano,azul), establece marciano=marty y la proposición triunfa.

Este proceso de abrirse paso a través de largas cadenas de reglas y plantearse a sí mismo una nueva proposición que demostrar a cada paso, es la forma en la que el PROLOG resuelve todos sus interrogantes. El método se denomina *retroceso* porque, si un camino determinado a través de las reglas no produce una respuesta, el PROLOG recorre el camino hacia atrás hasta llegar a un punto de opción anterior, y luego prosigue en esa nueva dirección. De esta forma, hallará su respuesta o bien explorará cada uno de los caminos posibles a través de las reglas. Por este motivo los programas en PROLOG se comprenden mejor como *árboles*, o cadenas de proposiciones, en vez de como listas de enunciaciones.

El programa Alvey

En la actualidad, el PROLOG desempeña un papel esencial en el programa de investigación Alvey para el desarrollo de "superordenadores" de la quinta generación. El programa se estableció en 1983 tras el informe del Alvey Committee, creado para aconsejar sobre las posibilidades de la investigación en colaboración como respuesta a la iniciativa tomada por los japoneses con su Proyecto de Ordenadores de la Quinta Generación. La cooperación es el factor primordial del proyecto Alvey: se estableció estrechamente vinculado con el proyecto ESPRIT (European Fifth Generation: quinta generación europea) y se han hecho contactos para establecer vínculos con los japoneses. Se han aprobado alrededor de 100 proyectos, todos los cuales responden a una estrategia de planificación central que se revisa y actualiza anualmente. Empleando las características del PROLOG de "base de datos inteligente", una máquina de este tipo se podría implementar directamente como un sistema experto. Otras aplicaciones del PROLOG que se están considerando actualmente incluyen su uso como parte de una interface para lenguaje natural. El PROLOG se presta particularmente para el proceso de lenguajes naturales y, dado que esta área es un aspecto esencial de las máquinas auténticamente "amables con el usuario" y fundamental para la investigación en materia de quinta generación, parece un hecho seguro que el PROLOG continuará ocupando un papel primordial en el desarrollo de nueva tecnología

Kowalski

Influencia paterna
Profesor Bob Kowalski, a quien se considera "padre de la programación lógica", cuyas ideas respecto a la programación y los sistemas lógicos utilizara A. Colmerauer para desarrollar en la Universidad de Marsella el primer intérprete de PROLOG

Ganar o perder

El diseño del algoritmo final de nuestro juego de simulación incluye la eventual participación en una revuelta local y los preparativos del viaje de regreso

Al entrar en la fase final del juego, ha concluido el intercambio comercial y se han cargado a bordo del barco las mercancías adquiridas al jefe de los nativos del Nuevo Mundo. A la mañana siguiente, la tripulación se reaprovisionará de agua fresca y alimentos e iniciará el viaje de regreso. Se han comerciado todas las mercancías, a excepción de las armas, y al regresar a puerto la venta de las especias, figurillas y perlas reportará algún beneficio.

Sin embargo, antes de emprender el viaje de regreso, se produce un nuevo acontecimiento. Durante la noche, un rival del jefe visita en secreto el barco. El insurgente desea comprarle sus armas con el fin de derrotar al jefe, y ofrece 30 perlas por cada una. Si usted acepta el trato obtendrá un inmenso beneficio, pero debe tener en cuenta algunos puntos:

Si el jefe descubre que usted ha vendido las armas, seguramente se vengará de alguna forma en caso de que la rebelión fracasase. No obstante, el margen de beneficios del viaje podría ser tan reducido que quizá usted considere que las ganancias extras bien valen el riesgo a correr. Puede, por supuesto, rechazar la oferta, en cuyo caso el jefe, como gesto de agradecimiento, le proporcionará provisiones gratuitamente para el viaje de regreso e incluso podría incluir 50 perlas como recompensa. Existe la posibilidad, si vende las armas, de que la revuelta triunfe; pero si fracasa, las consecuencias serán desastrosas. El jefe, tras enterarse de su doble juego, ordenará que se prenda fuego a su barco, con lo cual el juego terminará.

Si usted puede evitar este desastre final, el barco emprenderá el regreso a puerto; la tripulación se halla en buena forma, los vientos son favorables y el viaje se realizará en ocho semanas. Cuando el barco llegue a puerto, se pueden vender las mercancías, pagarle a la tripulación y, con suerte, obtener algún beneficio. Si no hubiera suficiente dinero para pagarle a la tripulación, se podría vender el barco para cubrir gastos. El informe final del programa calcula una hoja de balance para el viaje y visualiza información en el sentido de si ha terminado con pérdidas o ganancias.

El programa principal llama, en la línea 893, a una subrutina de la línea 10300, que se ocupa de la oferta hecha por el rival del jefe de adquirir sus armas. El programa comprueba primero si hay algún arma disponible para el intercambio, porque si no hubiera ninguna no tendría sentido que el insurgente contactara con el capitán. La línea 10305 examina el segundo elemento de la matriz de provisiones, OA(2), que corresponde a las armas, y si éste es igual a 0, no habrá, por supuesto, ningún arma y se devolverá el control al programa principal. Si hubiera armas disponibles, el insurgente formulará su oferta.

Se le indicará que digite S o N a la propuesta en la línea 10326, y la línea 10330 comprobará si la entrada es correcta. Si no lo es, retrocederá y solicitará otra respuesta.

Si decide vender las armas, la rutina saltará a la línea 10400, donde finalmente se podrá desarrollar el evento. La línea 10405 genera un número aleatorio entre 0 y 1, proporcionando un 75 % de probabilidades de enviar el programa a la línea 10450, que hará fracasar la rebelión. Si usted cree que las probabilidades en contra de que triunfe la revuelta son demasiado elevadas, puede variarlas para adecuarlas mejor a su sentido de la equidad.

Si la revuelta fracasa, el jefe se apropiará de las mercancías e incendiará el barco, con lo cual el juego habrá terminado. De lo contrario, el programa continuará a través de la línea 10429, que asegura el triunfo de la rebelión, y el nuevo jefe recientemente proclamado aprovisionará por completo el barco antes de que éste vuelva a hacerse a la mar.

La ecuación de la línea 10429 calcula la cantidad de perlas obtenidas en el intercambio, multiplicando la cantidad de armas, OA(2), por 30, sumándole el total subsiguiente al primer elemento de la matriz AO() para registrar la cantidad de perlas adquiridas. La línea 10430 suprime las armas de la matriz de provisiones. A los fines del juego no es estrictamente necesario hacer esto, pero es un buen hábito de programación y hará que resulte más sencillo ampliar el juego para incluir, por ejemplo, incidentes durante el viaje de regreso.

Si usted se niega a aprovechar el ofrecimiento, el jefe actual podría recompensarlo con 50 perlas. Esto se decide al azar en la línea 10345. La subrutina final comienza en la línea 10500 y se ocupa del viaje de regreso y la venta de las mercancías.

Complementos al BASIC

Spectrum

Reemplazar AO() por E(), V1() por B() y V2() por D() en todo el listado e introducir estos cambios:

```
10310 CLS:GO SUB 9200
10328 INPUT I$:LET I$=I$(TO 1)
10354 LET I$=INKEY$:IF I$="" THEN GO TO
      10354
10400 CLS:GO SUB 9200
10501 CLS:GO SUB 9200
10547 LET I$=INKEY$:IF I$="" THEN GO TO
      10547
```

BBC Micro:

Introducir las siguientes modificaciones:

```
10310 CLS: GO SUB 9200
10354 I$=GET$
10400 CLS:GOSUB 9200
10501 CLS:GOSUB 9200
10547 I$=GET$
```

Módulo 13: Fin del viaje

Adición al cuerpo principal del programa

```
893 GOSUB 10300
894 GOSUB 10500
```

Rutina Revuelta

```
10300 REM REVOLUCION
10305 IF OA(2)=0 THEN RETURN
10310 PRINT CHR$(147):GOSUB 9200
10315 S$="DURANTE LA NOCHE UN RIVAL DEL*":GOSUB 9100
10316 S$="JEFE VISITA EL BARCO EN SECRETO*":GOSUB 9100
10317 PRINT:GOSUB 9200
10318 S$="QUIERE COMPRAR TUS ARMAS*":GOSUB 9100
10320 S$="PARA UNA REVUELTA*":GOSUB 9100
10322 PRINT:GOSUB 9200
10324 S$="TE OFRECE 30 PERLAS POR CADA ARMA*":GOSUB 9100
10326 S$="LE VENDES LAS ARMAS?(S/N)*":GOSUB 9100
10328 INPUT I$:I$=LEFT$(I$,1)
10330 IF I$<>"N"AND I$<>"S" THEN 10328
10332 IF I$="S" THEN 10400
10334 PRINT:GOSUB 9200
10336 S$="EL JEFE SE ENTERA Y SE SIENTE*":GOSUB 9100
10338 S$="AGRADECIDO HACIA TI*":GOSUB 9100
10340 S$="TE DA PROVISIONES GRATIS*":GOSUB 9100
10342 S$="PARA EL VIAJE DE REGRESO*":GOSUB 9100
10344 GOSUB 9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="Y 50 PERLAS!!*":GOSUB 9100
10348 AO(1)=AO(1)+50
10350 PRINT:GOSUB 9200
10352 S$=K$:GOSUB 9100
10354 GET I$:IF I$=""THEN 10354
10359 RETURN
10400 PRINT CHR$(147):GOSUB 9200
10405 IF RND(1)<.75 THEN 10450
10410 S$="LA REVUELTA HA TRIUNFADO*":GOSUB 9100
10412 PRINT:GOSUB 9200
10415 S$="EL NUEVO JEFE TE RECOMPENSA CON*":GOSUB 9100
10420 S$="PROVISIONES GRATUITAS PARA EL VIAJE*":GOSUB 9100
10425 S$="DE REGRESO.*":GOSUB 9100
10429 AO(1)=AO(1)+(OA(2)*30):REM SUMAR PERLAS
10430 OA(2)=0
10431 GOTO 10350
10450 S$="LA REVUELTA FRACASA!!*":GOSUB 9100
10452 PRINT:GOSUB 9200
10455 S$="EL VIEJO JEFE ESTA INDIGNADO CONTIGO*":GOSUB 9100
10457 S$="TE QUEMA EL BARCO Y ROBA*":GOSUB 9100
10458 S$="TODO!!*":GOSUB 9100
10459 PRINT:GOSUB 9200
10460 S$="      JUEGO TERMINADO!!      ":GOSUB 9100
10462 END
10464 GOTO 10462
```

Rutina Fin del viaje

```
10500 REM FIN DEL VIAJE
10501 PRINTCHR$(147):GOSUB 9200
10505 S$="CON UN TRIPULACION FUERTE Y VIENTOS*":GOSUB 9100
10507 S$="FAVORABLES EL VIAJE DE REGRESO VA*":GOSUB 9100
10508 S$="BIEN Y DURA SOLO 8 SEMANAS*":GOSUB 9100
10512 WW=0
10514 FOR T=1 TO 5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT:GOSUB 9200
10520 S$="FACTURA SALARIAL PARA EL VIAJE DE REGRESO=*":GOSUB
      9100
10522 PRINT WW;"PIEZAS DE ORO"
10524 PRINT:GOSUB 9200
10526 S$="CUANDO REGRESAS*":GOSUB 9100
10528 S$="PARA VENDER TUS MERCANCIAS*":GOSUB 9100
10530 S$="ESTAS VALEN ENTONCES:*":GOSUB 9100
10532 PRINT"PERLAS-";V2(1);"PIEZAS DE ORO"
10534 PRINT"FIGURILLAS-";V2(2);"PIEZAS DE ORO"
10536 PRINT"ESPECIAS-";V2(3);"PIEZAS DE ORO"
10538 PRINT:S$="OBTIENES UN TOTAL DE*":GOSUB 9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*(V2(3))
10542 PRINT X;"PIEZAS DE ORO"
10545 PRINT:S$=K$:GOSUB 9100:PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="AHORA POSEES:*":GOSUB 9100
10552 PRINT MO+X;"PIEZAS DE ORO"
10555 PRINT:GOSUB 9200
10556 S$="LA FACTURA SALARIAL PARA EL VIAJE ES DE*":GOSUB 9100
10557 PRINT WT+WW;"PIEZAS DE ORO"
10559 PRINT:GOSUB 9200
10560 S$="TERMINAS EL VIAJE CON:*":GOSUB 9100
10562 Z=MO-X-WT-WW
10565 PRINT Z;"PIEZAS DE ORO"
10566 PRINT:GOSUB 9200
10567 PRINT:S$="TU CLASIFICACION ES:*":GOSUB 9100:PRINT
10568 IF Z>3200 THEN S$="CAPITALISTA DE PRIMER ORDEN*":GOSUB
      9100:END
10569 IF Z>2500 THEN S$="INSIGNE COMERCIANTE*":GOSUB 9100:END
10570 IF Z>2000 THEN S$="MERCACHIFLE DE III CLASE*":GOSUB 9100:END
10571 IF Z>1000 THEN S$="MAS BIEN UN PRIMO*":GOSUB 9100:END
10572 S$="MAS PATO QUE PIRATA"
10573 GOSUB 9100:END
```

El viaje de regreso dura ocho semanas y se debe calcular la factura de salarios. En la línea 10512 se crea una variable, WW, que representa la factura salarial para el viaje de regreso. Su valor se calcula en el bucle que empieza en la línea 10514. La fórmula multiplica el contador de cada categoría de tripulante, CC(T), por el salario semanal para cada uno, WG(T), para producir la factura salarial semanal para todos los tripulantes de esa categoría. El contador del bucle T asegura que este proceso se repita para cada categoría de tripulante, produciendo finalmente una factura de salarios semanal total para toda la tripulación. Para calcular la factura del viaje de regreso, este nuevo total se multiplicará por 8, y el programa imprimirá la factura total para la travesía de retorno.

Cuando el barco llegue a puerto se venderán las mercancías. Sus valores de mercado se establecen mediante la matriz V2(3) DIMensionada en la línea 62. Estos valores se establecieron al azar al principio del programa. Las líneas 10532-10536 de la rutina fin del viaje visualizan los valores retenidos en esta matriz, para cada uno de los tipos de mercancías comerciadas.

La cantidad total de oro recibida por las mercancías comerciadas se calcula mediante la línea 10540 y se almacena en X. La fórmula utilizada para este cálculo multiplica la cantidad de cada artículo, registrada en la matriz AO(), por el valor de mercado actual almacenado en V2(). Se suman los ingresos por las perlas, figurillas y especias y se imprime el total.

Es posible que al inicio del viaje no se haya gastado todo el capital original de 2 000 piezas de oro. Cualquier saldo se conservará a lo largo del viaje en la variable MO y se sumará al oro recibido durante la venta; la cantidad se visualizará mediante la línea 10552. La factura salarial total para el viaje se calcula en la línea 10557 sumándole la factura salarial para el viaje de ida, WI, a la factura para el viaje de regreso. El programa calcula las ganancias comerciales restando la factura salarial total de la cantidad de oro que haya en las arcas. Por último, se juzga lo bien que se ha desempeñado usted produciendo una clasificación basada en el monto de los beneficios que haya obtenido.

Ian McKinnell

El mejor lugar

Iniciamos una serie dedicada a analizar en detalle el sistema operativo del Sinclair Spectrum. Examinaremos, en primer lugar, el mapa de memoria

A diferencia del BBC Micro, el Spectrum de Sinclair no tiene una ROM dentro del ordenador reservada para el sistema operativo, puesto que un mismo chip alberga el intérprete del BASIC y el OS. Más aún, sólo existe una versión de esta ROM y el chip que la alberga es el mismo tanto en el Spectrum como en el Spectrum+, por lo que no se pueden esperar muchos cambios en el futuro. El OS es un conjunto orgánico de rutinas (de un tamaño la mitad que el OS del BBC) que ocupa tan sólo 7 K del chip del Spectrum que contiene la ROM de 16 Kbytes.

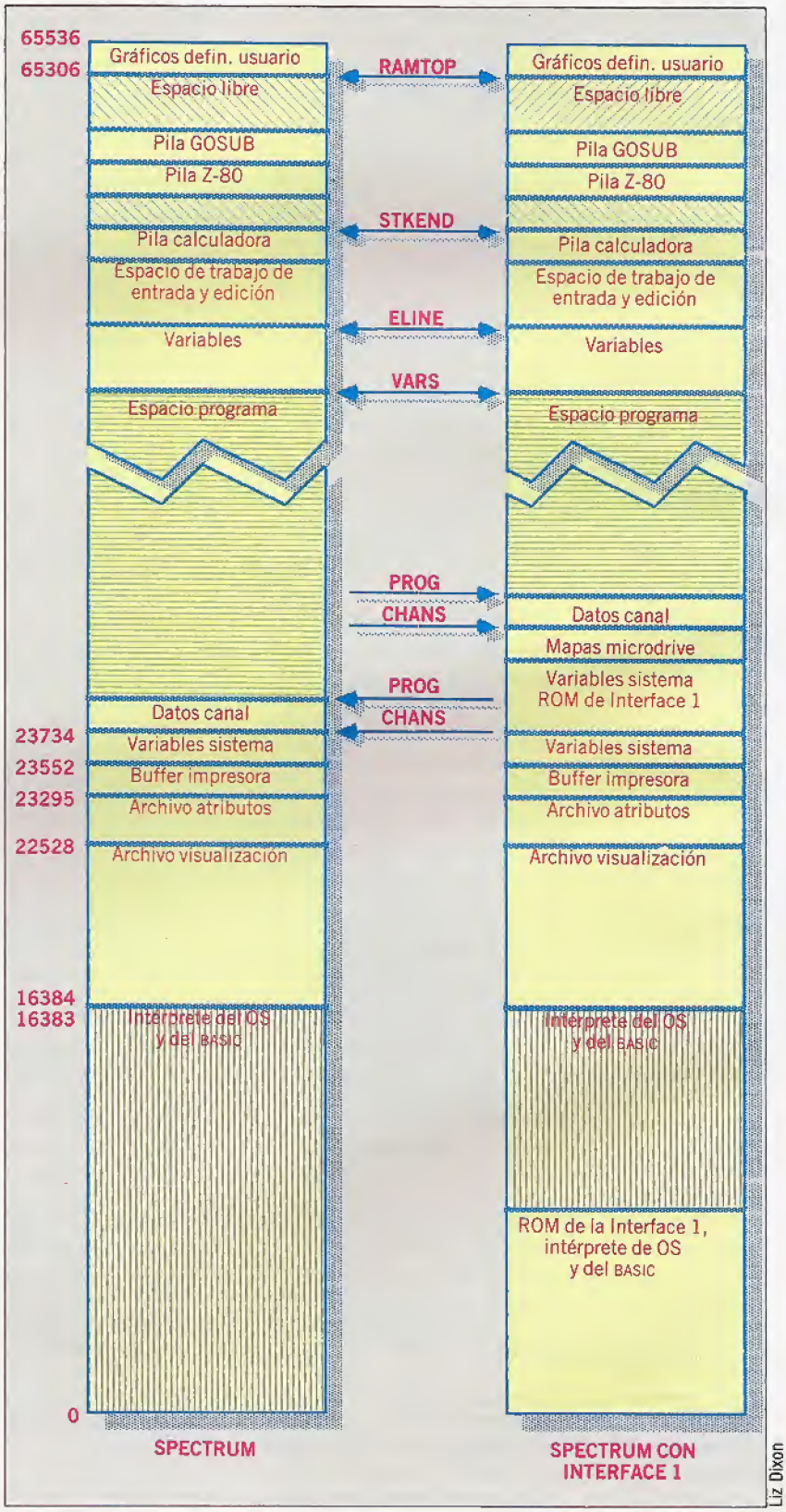
A pesar de la falta de variantes y de la naturaleza compacta de la codificación, el OS del Spectrum presenta una desventaja respecto del OS del BBC Micro. Salvo mínimas excepciones, no está vectorizado, por lo que resulta más difícil alterar el tratamiento que hace el OS de ciertos eventos. Difícil pero no imposible, como tendremos oportunidad de observar.

Suponiendo que usted estudió los capítulos referentes al OS del BBC Micro, en este momento tendrá una cierta familiaridad sobre el papel desempeñado por el sistema operativo en un ordenador. Como breve resumen, le recordamos que el OS se ocupa de todas aquellas rutinas de operaciones de entradas, salidas, visualización y almacenamiento en ficheros, interrupciones, y, en general, de toda operación que haga de puente entre el hardware y el programa del usuario o el intérprete del BASIC. Comencemos analizando el mapa de memoria, que vemos ilustrado a la derecha.

El primer bloque de 16 K de la memoria está ocupado por un chip de ROM que contiene el intérprete del BASIC y el sistema operativo. Grosso modo, el OS ocupa los 6 o 7 K inferiores de la ROM, y el intérprete, la parte superior restante. Cuando se emplea una Interface 1, los 8 K inferiores de la memoria se distribuyen en páginas de un modo similar a la paginación ya vista en el BBC Micro realizada en el área de memoria que va desde &8000 al &BFFF. Cuando se desea acceder a alguna de las facilidades que ofrece la Interface 1 (microdrives, interface serial, red de área local) la ROM normal queda "despaginada" y la ROM de la Interface 1 "paginada". Pero de esto hablaremos detalladamente más adelante.

El resto del espacio de la memoria lo ocupa la RAM, y es de notar que existen muy pocas diferen-

Datos internos
Estos mapas de memoria de un Spectrum (o de un Spectrum+) ilustran cómo afecta a ciertas áreas de la RAM el acoplamiento de una Interface 1. El área para variables de sistema se extiende normalmente hasta 23733, pero supera esta posición cuando existe una Interface 1



cias en el empleo de la RAM de un Spectrum cuando se está utilizando la Interface 1.

Si examinamos cada una de las secciones del mapa, el *archivo de la visualización (display file)* almacena información sobre la forma de las figuras que han de visualizarse en la pantalla. Los datos referentes al color, BRIGHT y FLASH no están en este archivo, sino que se almacenan en el *fichero de atributos (attribute file)*, ya que esta información corresponde a un cuadrado de carácter en la pantalla. El *buffer para impresora (printer buffer)* es un área de 256 bytes que requiere la impresora ZX cuando se desea una copia sobre papel de la pantalla.

Las *variables del sistema* son empleadas por el OS y por el intérprete, de modo que puedan inspeccionar lo que está sucediendo en el sistema. Más adelante estudiaremos algunas de las variables del sistema más imprescindibles. En el mismo manual para usuarios del Spectrum se encuentra una lista completa de tales variables. Lista que, por desgracia, falta en el manual del Spectrum+.

Aquí es donde se hacen mucho más evidentes las diferencias entre un Spectrum con o sin Interface 1. Es evidente que la ROM de la Interface 1 necesita un espacio de trabajo y variables del sistema para sus tareas, de modo que cuando está siendo empleada genera algunas variables extra, que son colocadas a continuación de las variables del sistema normales.

Esto produce un efecto de "corrimiento" de toda la memoria empleada, con lo cual hay que realizar ciertos reajustes en consonancia.

Los *mapas para microdrive (microdrive maps)* son asimismo típicos de la Interface 1, empleados tan sólo cuando ésta vigila las áreas ocupadas y no ocupadas de la cinta del microdrive.

El área para *datos de canal (channel data)* nos ofrece una forma reducida de vectorización para las instrucciones LLIST, LPRINT, INPUT# y PRINT#. Así, por ejemplo, podemos escribir código máquina para dirigir una impresora corriente y alterar los datos de canal para que cuando encuentre la instrucción LPRINT o LLIST obedezca a nuestro código máquina en vez de las habituales rutinas del intérprete del BASIC. También de esto hablaremos con mayor extensión.

Los datos de canal también son empleados por la Interface 1.

Llegamos ahora al espacio de trabajo del programa en BASIC y de las variables, que el sistema emplea para almacenar el texto de un programa en BASIC y las variables requeridas por éste.

Los espacios de trabajo para edición *(edit)* y entrada *(input)* de datos sirven para obtener mediante EDIT una línea entera de programa o para entrar datos y líneas de programas. Una vez finalizada la edición o la entrada del texto, las líneas de programa se almacenan en su lugar específico dentro del programa, quedando libres estas áreas de nuevo.

Encontramos después un grupo de pilas. La *pila calculadora* sirve al intérprete del BASIC para sus operaciones aritméticas. La *pila Z80* es utilizada por la CPU de modo muy semejante a como el 6502 emplea la página uno de su memoria, es decir, le sirve para almacenar direcciones de retorno con subrutinas en código máquina y datos que han sido llevados a la pila para su almacenamiento temporal. La *pila* GOSUB es otra de las partes de la memoria que utiliza el intérprete del BASIC. Almacena las direcciones de retorno en las sentencias GOSUB del BASIC.

Por último, el área para gráficos *definidos por el usuario* retiene los datos de los caracteres que fueron, claro está, definidos por éste.

Dónde colocar el código máquina

De cuanto se ha dicho hemos de concluir que no hay reservado en el mapa de la memoria espacio alguno para los programas en código máquina. Es lo que sucede también en el BBC Micro si se le acopla una unidad de disco o una interface Econet. Tres son las áreas posibles para colocar en el Spectrum el código máquina. Están en una sentencia REM de línea 1 (se puede averiguar la posición de la línea 1 en la memoria a través del valor retenido en la variable del sistema PROG). Si a este valor sumamos cinco obtenemos la dirección del primer carácter en una sentencia REM de línea 1.

Otras áreas son el buffer para impresora y entre RAMTOP (parte superior de la RAM) y la parte de memoria usada para almacenar datos de gráficos definidos por el usuario. Este último método es similar al empleo de la memoria entre HIMEM y el inicio de la RAM de video en el BBC Micro.

El cuadro que proporcionarnos a continuación compara los métodos entre sí:

Sitio	Spectrum	Spectrum e Interface 1
REM línea 1	✓	✗
Buffer impresora	(✓)*	(✓)*
Encima de RAMTOP	✓	✓

* No utilizable cuando se emplea una impresora

Vemos que sólo la última posibilidad es utilizable en la práctica, dado que los restantes métodos son arriesgados en determinadas circunstancias. Lo que se encuentra por encima de RAMTOP queda a salvo de una posible sobreescritura del BASIC, además de ser fácil la reserva de espacio. La posición exacta de RAMTOP se retiene como una variable de dos bytes en las direcciones 23730 y 23731. El valor se almacena en el formato corriente del Z80 *(lo-hi)*. Para comprobar la posición actual de RAMTOP haga lo siguiente:

```
LET ramtop=PEEK23730+256*PEEK23731
```

Una vez anotada la posición de RAMTOP (que suele ser 65306 en un ordenador de 48 K "limpio"), se habrán de calcular cuántos bytes se necesitan para el programa en código máquina. Después de calcular el valor que se obtiene mediante la fórmula RAMTOP−(n.º de bytes en el programa+1), se bajará RAMTOP con la instrucción CLEAR:

```
CLEAR valor
```

Esta instrucción, en un programa en BASIC, realizará un CLS, repondrá la posición de PLOT y ejecutará un RESTORE. Limpiará, además, y reubicará la pila GOSUB. Por esta razón habrá de habituarse a emplear con cuidado la instrucción y a colocarla en lo posible al principio del programa, de lo contrario no evitará los problemas.

Después de ejecutar la instrucción CLEAR valor, la

nueva posición de RAMTOP estará almacenada en 23730 y 23731, y el primer byte disponible para su código máquina se encontrará en la dirección (valor+1).

Así, suponiendo una máquina de 48 K limpia, la orden CLEAR 59999 reservará unos 5 K de memoria para sus programas, comenzando por la posición 60000.

Tras la reserva de memoria anterior, la instrucción POKE puede servir para almacenar los bytes que componen su programa en la memoria.

Variables del sistema

Finalizaremos esta introducción sobre el OS del Spectrum repasando algunas variables del sistema que contienen información sobre el modo como la memoria está distribuida en el sistema. El acceso a dichas variables ha de realizarse mediante PEEK o bien POKE referidas a las posiciones adecuadas, pues no existen equivalentes de OSBYTE u OSWORD, llamadas que analizamos detalladamente al estudiar el sistema operativo del BBC Micro.

- chars: Esta variable, situada en 23606 y 23607, apunta a una posición situada 256 bytes más abajo de la información del conjunto de caracteres. Habitualmente la dirección que alberga esta variable de dos bytes apunta a la dirección que está en la ROM 256 bytes más abajo respecto a la dirección del primer byte de la definición del carácter espacio (CHR$32).

La dirección actualmente albergada en la variable puede conocerse, como la de las restantes, de la siguiente manera:

```
LET chars=PEEK 23606+256*PEEK 23607
```

Si se desea, es posible alterar el valor contenido en la variable para que apunte a la información que defina un nuevo conjunto de caracteres, que puede almacenarse en la RAM.

- vars: Se encuentra en las posiciones 23627 y 23628, y contiene la dirección de inicio de las variables del BASIC. Se pueden inspeccionar (PEEK) estos valores pero no alterarlos, ya que "despistaría" al Spectrum sobre sus propias variables. La línea:

```
LET vars=PEEK 23627+256*PEEK 23628
```

le proporciona la dirección de inicio de las variables.

- prog: Contiene la dirección de inicio del programa en BASIC.

Es, pues, equivalente a la variable de sistema PAGE en el BBC Micro. Se halla en 23635 y 23636, por lo que si se desea conocer la dirección del comienzo del programa, haremos

```
LET prog=PEEK 23635+256*PEEK 23636
```

Es de notar que en este caso

```
PRINT vars-prog
```

nos dará la longitud del programa.

- eline: Se encuentra en 23641 y 23642. Contiene el inicio de cualquier texto entrado en el sistema. Se puede emplear para conocer la cantidad de memoria ocupada por las variables empleadas. Suponiendo calculada vars, escribiríamos

```
LET eline=PEEK 23641+256*PEEK 23642
PRINT eline-vars
```

para obtener dicha cantidad de memoria.

- chans: Contiene la dirección del inicio de la información del canal. Situada en 23631 y 23632, estudiaremos con más detalle el tema de los canales en próximos capítulos.

- stkend: Se encuentra en las posiciones 23653 y 23654. Junto con el valor de RAMTOP, nos permite saber la cantidad de memoria libre para nuestro programa en BASIC y sus variables. Escribiremos, para obtener el valor de stkend:

```
LET stkend=PEEK 23653+256*PEEK 23654
```

Y para conocer la cantidad de memoria que queda:

```
PRINT ramtop-stkend
```

También es posible conocer dicha cantidad mediante una rutina de la ROM. Pero este método no es tan exacto como el que acabamos de proponer. Suponiendo calculado el valor de RAMTOP:

```
PRINT ramtop-USR 7962
```

nos calculará también el espacio libre.

Hora del assembly

Hay varios paquetes ensambladores para el usuario del Spectrum interesado en profundizar en el código máquina. Uno de los más conocidos es DEVPAC 3, de Hisoft. Distribuido en cassette, incluye el ensamblador GENS3 y el monitor/desensamblador MON3. Existe también la versión para microdrive. Aunque su empleo está muy extendido, no es de los más cómodos del mercado, aunque también es verdad que resulta el más barato.

El Editor Assembler y el Spectrum Monitor (que incluye un desensamblador), de Picturesque, es un paquete doble, el primero a un precio algo más elevado que el segundo. Una de las principales ventajas del sistema de Picturesque es que se puede acoplar muy bien a un Spectrum de 16 Kbytes. Los manuales que acompañan a este paquete superan al de Hisoft, y el conjunto tiene un carácter mucho más profesional

Chris Stevens

Vara para medir

Una facilidad ausente en el Spectrum es el medio para calcular la longitud de un programa; esto se consigue con PEEK, pero aquí va una pequeña utilidad que se sirve de las variables PROG y VARS para calcular la longitud de cualquier programa en BASIC actualmente en la memoria, y la almacena en la REM de línea 1. Sólo exige que la primera línea del programa sea una sentencia REM de al menos cinco caracteres, como, por ejemplo:

```
1 REM 00000 bytes de largo
```

cuando se ejecuta la rutina en código máquina, la sentencia REM sufre una modificación para contener la longitud del programa. Por ejemplo:

```
1 REM 00801 bytes de largo
```

Esto es bastante útil, ya que a la siguiente vez que se examine el programa tendremos presente su longitud. El primer fragmento de código es un listado en assembly del código máquina en cuestión. Si no posee un ensamblador conveniente, el segundo listado está en BASIC y es un programa utilizable en la carga del programa en código máquina. En el Cargador de BASIC, CLEAR 61999 da ese número como valor de la variable de sistema RAMTOP. Por tanto, el primer byte "libre" se encuentra en la dirección 62000, que hemos empleado como dirección de inicio de la rutina en código máquina. RAMTOP tendrá este valor hasta que no venga alterado con una instrucción CLEAR n,o bien se ejecute una PRINT USR 0 e incluso si el ordenador se apaga y vuelve a encender. Esto quiere decir que NEW no afectará al valor de REMTOP, además de que la rutina en código máquina está protegida de cualquier sobreescritura en BASIC mientras RAMTOP quede inalterada. Una vez que el programa está en la memoria, se emplea USR para ejecutar el código máquina. El argumento de la función USR (n) es la dirección de la rutina en lenguaje máquina que desea ejecutar. Tal dirección puede estar en cualquier sitio en la memoria. Así, lo siguiente ejecutará el código máquina en la dirección 62000, y cualquiera puede ser usado para ejecutar el programa en código máquina que acabamos de ver:

```
RANDOMISE USR (62000)
PRINT USER (62000)
LET L=USR (62000)
```

Los métodos con el PRINT y el LET con USR dan también resultados utilizables por el BASIC. El resultado es el valor contenido en el par de registros BC cuando la rutina en código máquina termina y retorna al BASIC; así, una rutina en código máquina puede devolver los resultados al BASIC. Si su rutina no afecta al valor del registro BC, lo que se obtiene es la dirección de la rutina.

Listado assembly

```
62000                   10          ORG   62000
62000  2A4B5C           20          LD    HL,(23627)   toma VARS en HL
62003  ED5B535C         30          LD    DE,(23635)   toma PROG en DE
62007  AF               40          XOR   A            limpia flag arrastre
62008  ED52             50          SBC   HL,DE        pone longitud progr. en HL
62010  DD2A535C         60          LD    IX,(23635)   inicio programa en IX
62014  DD23             70          INC   IX
62016  DD23             80          INC   IX           incrementa IX para que apunte
62018  DD23             90          INC   IX           al carácter post. al token REM
62020  DD23             100         INC   IX           de la primera línea del prog.
62022  DD23             110         INC   IX
62024  011027           120         LD    BC,10000     inicio toma códigos ASCII
62027  CD6BF2           130         CALL  PRINT        de cada dígito de la long. del
62030  DD23             140         INC   IX           programa y lo almacena en
62032  01E803           150         LD    BC,1000      la sentencia REM
62035  CD6BF2           160         CALL  PRINT
62038  DD23             170         INC   IX
62040  016400           180         LD    BC,100
62043  CD6BF2           190         CALL  PRINT
62046  DD23             200         INC   IX
62048  010A00           210         LD    BC,10
62051  CD6BF2           220         CALL  PRINT
62054  DD23             230         INC   IX
62056  010100           240         LD    BC,1
62059  AF               250 PRINT:  XOR   A            limpia cont. arrastre y cero
62060  B7               260 L1:     OR    A
62061  ED42             270         SBC   HL,BC        resta potencia de diez de
                                                       la long. del programa...
62063  3803             280         JR    C,FINISH     y salta si es negativo
62065  3C               290         INC   A            si no lo es incrementa A
62066  18F8             300         JR    L1           repetición
62068  09               310 FINISH:ADD   HL,BC        repone la long. del prog.
                                                       a valor positivo para
                                                       la siguiente potencia de diez
62069  C630             320         ADD   A,48         toma el código ASCII del dígito
62071  DD7700           330         LD    (IX),A       lo almacena en REM
62074  C9               340         RET                fin
```

Cargador del BASIC

```
  10 REM 00000 bytes de largo
  20 CLEAR 61999
  30 FOR i=0 TO 74
  40 READ a: POKE 62000+i,a
  50 NEXT i
1000 DATA 42, 75, 92, 237, 91, 83, 92, 175, 237, 82, 221, 42, 83, 92, 221, 35,
     221, 35, 221, 35, 221, 35, 221, 35, 1, 16, 39, 205, 107, 242, 221, 35, 1,
     232, 3, 205, 107, 242, 221, 35, 1, 100, 0
1010 DATA 205, 107, 242, 221, 35, 1, 10, 0, 205, 107, 242, 221, 35, 1, 1, 00,
     175, 183, 237, 66, 56, 3, 60, 24, -8, 9, 198, 48, 221, 119, 0, 201
```

Nota: Para asegurarse de que esta *Vara para medir* localiza la posición correcta para los dígitos indicadores de la longitud del programa, no se inserten espacios entre el número de la primera línea y la instrucción REM (fuera del espacio automático que prevé el sistema del Spectrum)

Chris Stevens

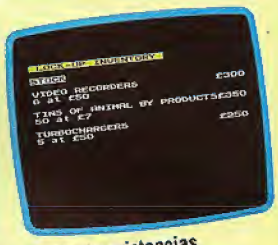
De la TV a la VDU

Haciendo su agosto
Estas son cuatro escenas tomadas de Minder. El objetivo del juego es vender su propio stock (almacenado en su "calabozo") a los comerciantes. Si usted se queda sin existencias, puede adquirir otras mercancías a través de los clientes del bar Winchester. Por supuesto, debe regatear para conseguir el mejor precio posible, tanto a la hora de comprar como de vender, con el fin de ganar la máxima cantidad de dinero en el tiempo de que dispone.

El bar Winchester

Haciendo un negocio

El "calabozo"



Lista de existencias

"Minder" (Guardián), programa producido por la firma Euston Films, es un original y entretenido juego inspirado en una popular serie de la televisión británica

El programa de televisión *Minder* se ha convertido en uno de los mayores éxitos de la televisión británica de los últimos años, y su conversión al formato de juego quizás haya llegado con demasiado retraso (la serie comenzó a emitirse a mediados de los setenta).

Minder se basa en los oscuros asuntos de Arthur Daley, conocido traficante de mercancías dudosas de todo tipo, y Terry McCann, su *minder* (guardián) y secuaz. En este juego usted representa el papel de Arthur, y el objetivo consiste en ganar el máximo dinero posible en quince días mediante la compra y venta de mercancías. Con este fin, debe contactar con otros comerciantes a quienes vender su mercancía. Con frecuencia tendrá cosas bastante peculiares que vender, como latas de subproductos animales o "ferkinators nucleónicos". Con igual frecuencia, los comerciantes le ofrecerán una cantidad demasiado reducida por los artículos y usted habrá de regatear con ellos para llegar a un acuerdo más razonable.

Una de las mayores espinas de Arthur es el sargento-detective Chisholm, oficial del Departamento de Investigación Criminal local. La influencia de Chisholm comienza cuando un comerciante interesado en comprarle algunos ordenadores personales, que *por casualidad* usted ha adquirido, le informa que el sargento-detective en realidad está a la caza de ordenadores robados. Estas noticias, por supuesto, hacen que el precio descienda.

Los problemas de Arthur se agravan aún más si Chisholm se presenta en el "calabozo" (*lock-up*: el almacén donde Arthur guarda sus mercancías) y encuentra los artículos robados.

Para hacer negocios y encontrar a Terry, a quien usted necesita para transportar sus mercancías desde y hasta el calabozo, por lo general habrá de acudir a Winchester, un club de bebedores. En este escenario, como en todos los otros del juego, hay hasta seis personajes con los que puede hablar. Sus rostros aparecerán en uno de los marcos de alrededor de la pantalla, y para abordar a uno de ellos usted simplemente pulsa la tecla correspondiente a ese personaje.

Abordar a los personajes de Winchester no es realmente necesario, a menos que desee hablar con alguno en particular, como Dave, el barman, o Terry; la mayoría de los demás intentarán venderle mercancías tan valiosas como jaulas de caoba para conejos o trajes de polietileno para la lluvia.

Aquí la estrategia es muy similar a cuando se trata de vender. Primero debe averiguar a cuánto está vendiendo estos artículos su contacto y qué cantidad de los mismos posee, y después puede empezar a regatear. Mientras negocia el precio, un reloj va marcando el tiempo: el contacto se marchará si el trato no se cierra dentro de un cierto margen. Si usted decide que ya no quiere regatear más, puede digitar BYE y esa persona desaparecerá.

Terry es un personaje algo esquivo y quizá no pueda establecer contacto con él durante algún tiempo, y cuando termina una tarea espera un favor a cambio: un trago o un premio. Por consiguiente, cuando negocie un trato debe tener en cuenta que de los beneficios habrá de extraer la paga de Terry.

Sin duda alguna, la parte más divertida de este juego es la negociación. Mientras usted y su oponente regatean el precio, la pantalla visualizará una serie de comentarios como "fantástica calidad" o "pequeño y simpático negociante". Cuando usted ofrece un precio, es importante que lo digite con la sintaxis correcta, como "Estoy ofreciendo 200 pesetas". De lo contrario, quizá el ordenador no le entienda.

A diferencia de muchos juegos de esta clase, *Minder* es tan entretenido e interesante como la serie de televisión. Consigue captar con exactitud la tónica general del programa y se hará popular tanto entre los amantes de los juegos como entre quienes conozcan la serie de televisión.

Minder: Para el Sinclair Spectrum, MSX, Memotech, Amstrad y Commodore 64
Editado por: DK'tronics, Shire Hill Industrial Estate, Saffron Walden, Essex CB11 3AQ, Gran Bretaña
Autor: Don Priestley
Formato: Cassette
Palanca de mando: No es necesaria

Sistemas expertos

Continuando con nuestra serie, centraremos nuestra atención en los programas denominados "sistemas expertos"

Un experto (un consejero hospitalario, un geólogo o un analista químico, p. ej.) suele ser una persona respetada. Los expertos dedican mucho tiempo a estudiar y practicar la disciplina elegida para llegar a ejercer eficientemente su trabajo. La utilización de expertos humanos presenta, no obstante, algunos problemas por su escaso número, su falibilidad, el pago de sus remuneraciones y, por supuesto, su calidad de mortales, que implica, a la larga, la pérdida definitiva de la destreza adquirida. Por este motivo es comprensible que a muchas empresas les atraiga la idea de codificar la pericia en programas de ordenador, para beneficiarse de poder trabajar con un gran cuerpo de conocimientos sin los inconvenientes propios de los expertos humanos.

El concepto de *sistema experto* nació en los años setenta, cuando los investigadores en el campo de la inteligencia artificial abandonaron, o pospusieron, la creación de máquinas inteligentes a nivel general y volcaron su interés en la solución de problemas del mundo real centrados en aspectos muy concretos. Por consiguiente, el sistema experto es uno de los primeros ejemplos de AI aplicada, y las técnicas para sistemas expertos se han extendido mucho más allá de los confines de los laboratorios de investigación en los que fueron concebidas. De hecho, en cierto modo los sistemas expertos han llevado a la AI al uso práctico cotidiano. Existen ya sistemas que superan a los seres humanos capacitados en diagnóstico médico, interpretación de espectrogramas de masa, clasificación de enfermedades de cultivos y otras muchas cosas más. Es interesante preguntarse, en consecuencia, cómo funcionan.

Típicamente, un sistema experto se basa en un amplio cuerpo de conocimientos sobre un área problemática específica. En general, este conocimiento se organiza como un conjunto de reglas que permitan que el sistema extraiga conclusiones a partir de datos o premisas dadas, capacitándolo, en consecuencia, para ofrecer un consejo inteligente o tomar decisiones inteligentes. Este enfoque al diseño de sistemas basado en el conocimiento representa un cambio evolutivo en la ciencia de los ordenadores, con consecuencias revolucionarias. Sustituye a la tradicional fórmula de *datos + algoritmo = programa* por una nueva arquitectura centrada alrededor de una *base de conocimientos* y un *motor de inferencias*, de modo que *conocimientos + inferencia = sistema experto*. Esta fórmula es, obviamente, similar, pero con un enfoque lo suficientemente diferente como para tener profundas implicaciones.

Marcus Wilson-Smith

En busca de consejo
Los sistemas que pueden adquirir conocimientos de expertos y utilizar esos conocimientos para ofrecer consejo o diagnósticos están adquiriendo creciente popularidad en muchas disciplinas, desde la medicina hasta la agricultura y la arquitectura. Los sistemas expertos, diseñados para operar en un campo temático muy delimitado, ofrecen a los profesionales un servicio de consulta en línea para ayudarles en su trabajo

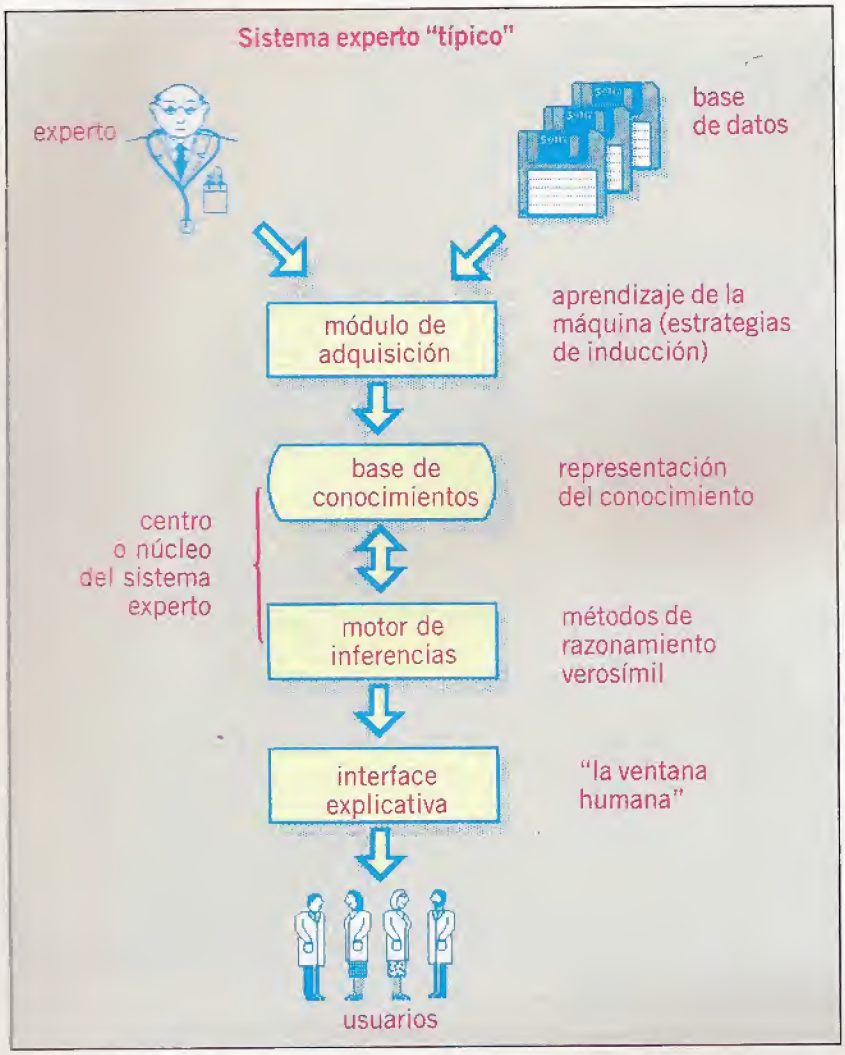
Para saber qué es un sistema experto resulta útil la siguiente lista de verificación de características típicas:

- Un sistema experto se limita a un campo de pericia relativamente delimitado.
- Debe ser capaz de razonar con datos inciertos y reglas no fiables.
- Debe ser capaz de explicar su cadena de razonamiento de una forma exhaustiva.
- Los hechos y los mecanismos de inferencia son "separables": el conocimiento no está codificado como parte de los procedimientos deductivos.
- Está diseñado para crecer por incrementos.
- Típicamente se basa en reglas.
- Su salida es un consejo o sugerencia, no tablas de cifras ni gráficos.

La palabra clave es "conocimiento". Está claro que el objetivo de un sistema inteligente para resolver problemas es omitir la búsqueda ciega o aleatoria. Para hacerlo, un sistema de ordenador ha de explotar la misma ventaja que tiene el experto humano en relación al novato, es decir, la pericia o conocimiento organizado: el conocimiento de hechos, de reglas de inferencia y de estrategias de solución. Un sistema experto totalmente viable tiene cuatro componentes esenciales:

1. La base de conocimientos.
2. El motor de inferencias.
3. El módulo de adquisición de conocimientos.
4. La interface explicativa.

Ian McKinnell



Sinopsis del sistema
Un sistema experto se compone de varios módulos que permiten que el conocimiento pase del experto al usuario final. En primer lugar, el conocimiento se debe adquirir del o de los expertos e incorporar a una base de conocimientos. Para poder hacer predicciones, dar consejo o proporcionar un diagnóstico, el sistema ha de ser capaz de extraer inferencias de la base de conocimientos. Por último, la interface explicativa permite que el usuario se comunique con el sistema con el fin de consultarlo

Los cuatro módulos son críticos. Un sistema basado en el conocimiento puede carecer de alguno de ellos, pero un auténtico sistema experto no puede carecer de ninguno. Hablaremos de estos cuatro módulos de uno en uno y explicaremos cómo trabajan todos juntos.

La base de conocimientos

Los dos componentes fundamentales de un sistema experto son la base de conocimientos y el motor de inferencias. La base de conocimientos almacena información relativa al dominio del tema; sin embargo, la información de una base de conocimientos no es el conjunto pasivo de registros y elementos que usted puede esperar hallar en una base de datos convencional. En cambio, contiene representaciones simbólicas de las reglas de juicio y experiencia de los expertos de una forma que permita al motor de inferencias extraer deducciones lógicas a partir de ellas.

La mayoría de los elementos de una base de conocimientos son no matemáticos. Las dos dificultades fundamentales al desarrollar una base de conocimientos son la representación del conocimiento y la adquisición de conocimientos. El primer problema se refiere a la decisión de cómo codificar el conocimiento de modo tal que el ordenador pueda utilizarlo. En general, se han de representar los elementos siguientes: términos del dominio (la jerga que emplean los expertos de ese campo), relaciones estructurales (las interrelaciones de entidades de los componentes) y relaciones causales (las relaciones causa-efecto entre los componentes).

La tarea del ingeniero de conocimiento consiste en seleccionar los medios adecuados para almacenar simbólicamente tal información. Se han desarrollado cuatro métodos principales:

- Reglas en formato IF...THEN. La condición especifica algún patrón y la conclusión puede ser una acción o aserción.
- Redes semánticas. Éstas representan relaciones entre objetos del dominio (p. ej., la ballena es un mamífero) mediante enlaces entre nudos.
- Marcos. Son estructuras de registros generalizadas que pueden tener valores por defecto y pueden tener codificadas acciones como los valores de ciertos campos o ranuras.
- Cláusulas "trompeta". Ésta es una forma de lógica de predicado en la cual se basa el PROLOG y con la cual el sistema PROLOG lleva a cabo inferencias.

Los primeros sistemas expertos utilizaban casi exclusivamente el formalismo basado en reglas. Una regla de muestra del sistema Mycin para el diagnóstico de infecciones de la sangre es típica de la estructura IF...THEN:

IF:
1. La infección que requiere una terapia es meningitis, y
2. El tipo de infección es fungal, y
3. No se observaron organismos en la solución colorante del cultivo, y
4. El paciente no es un huésped comprometido, y
5. El paciente ha estado en una región donde los coccidiomycosos son endémicos, y
6. La raza del paciente es negra de Asia o India, y
7. El antígeno cryptococal en el csf no fue positivo

THEN
Existen indicios que sugieren que el cryptococo no es uno de los organismos que podrían estar produciendo la infección.

A partir de este ejemplo podemos ver que un sistema experto emplea la jerga técnica del área en el cual fue diseñado para operar; en este caso, la medicina. La construcción IF...THEN utilizada por el Mycin es, esencialmente, una serie de sentencias que se pueden determinar como falsas o verdaderas. Por consiguiente, las sentencias se pueden enlazar mediante operadores booleanos, como AND, para ayudar a su tratamiento por el ordenador. Con el fin de extraer la información requerida para realizar un diagnóstico, el Mycin debe establecer un diálogo con el usuario del sistema. Obviamente, al menos en este sistema, el usuario ha de poseer un cierto nivel de conocimientos en la materia, de modo que pueda comprender las preguntas del sistema experto y responder a ellas.

Los mecanismos de inferencia consisten en métodos de búsqueda o razonamiento que permiten al sistema hallar soluciones y, de ser necesario, proporcionar justificaciones a sus respuestas. Existen dos estrategias de razonamiento globales: el encadenamiento hacia adelante y el encadenamiento hacia atrás.

El encadenamiento hacia adelante implica trabajar hacia adelante a partir de la evidencia (o síntomas) hacia las conclusiones (o diagnósticos). En

un sistema basado en reglas, simplemente implica emparejar las condiciones IF a los hechos, posiblemente por un orden predeterminado. El encadenamiento hacia adelante es fácil de informatizar y adecuado para casos en los que de alguna manera se han de reunir todos los datos. Ejemplos de tales casos son aquellos en los cuales los datos se generan automáticamente mediante un instrumento y en los cuales se ha de rellenar un formulario.

El encadenamiento hacia atrás trabaja desde la hipótesis hasta la evidencia. El sistema elige una hipótesis y busca datos para demostrarla o refutarla. Se puede programar de una forma recursiva y en los sistemas de estilo consultivo conducen típicamente a una clase de diálogo más natural. El problema de qué hipótesis seleccionar en cualquier situación dada aún no está resuelto del todo y, en consecuencia, en la práctica la mayoría de los sistemas emplean una mezcla de encadenamientos hacia adelante y hacia atrás.

Los expertos son notoriamente conocidos por su incapacidad para expresar cómo llegan a sus conclusiones, no necesariamente porque no deseen divulgar secretos sino porque muchos de sus procesos mentales se hallan por debajo del nivel de la consciencia, a nivel intuitivo. De modo que la adquisición de conocimientos ha llegado a considerarse como el principal cuello de botella en el desarrollo de sistemas expertos. Los expertos tienden, no obstante, a ser muy eficientes como críticos. Pueden analizar un caso tomado como ejemplo y decir qué decisión habrían tomado ellos y, si se les solicita, criticar la solución sugerida por un ordenador. Recientemente, por tanto, se ha dedicado mucha atención a desarrollar herramientas de software que permitan a un sistema experto inducir su propio conocimiento a partir de ejemplos previamente clasificados. En efecto, este poder contribuye al proceso de adquisición de conocimientos, superando al mismo tiempo muchas de las dificultades que entraña extraer conocimientos de los expertos humanos y la laboriosa tarea de codificarlos para el ordenador. Aun cuando el sistema no puede hacer todo el trabajo (desde una base de datos de casos a un conjunto de reglas de decisión) por sí mismo, parece útil que pueda refinar su propia base de conocimientos durante un periodo de "noviciado" o durante el uso, aprendiendo a partir de sus errores. En próximos capítulos analizaremos algunos sistemas de aprendizaje de las máquinas.

La interface explicativa

Una de las cosas buenas del Mycin, el Prospector y muchos sistemas expertos clásicos era que, si se les pedía, podían justificar sus conclusiones. Obviamente, cuando se está poniendo en manos del ordenador la responsabilidad de decisiones de vida o muerte, o incluso de pérdidas o beneficios, es esencial que el sistema explique su razonamiento. He aquí un ejemplo de cómo responde Mycin a una pregunta WHY? (por qué) formulada por su usuario (que es un médico, no un paciente, y no aceptará diagnósticos a ojos cerrados sin más explicación). El diálogo comienza con Mycin solicitando información.

¿Es la meningitis una infección adquirida en el hospital?
WHY?

[2.0] Esto ayudará a determinar los organismos (distintos a aquellos observados en cultivos o frotis) que podrían estar causando la infección. Ya se ha determinado que:
[2.1] la infección que requiere terapia es meningitis, y
[2.2] los organismos no se observaron en la solución colorante de este cultivo csf pendiente, y
[2.3] el tipo de infección es bacteriana, y
[2.4] la neurocirugía no es uno de los elementos relevantes a partir de la historia clínica de este paciente, y
[2.5] la edad de este paciente es de más de 10 días.
Por lo tanto, si:
[2.6] la infección se adquirió mientras el paciente estaba hospitalizado,
entonces existen indicios de que los organismos (distintos a aquellos observados en cultivos y frotis) que podrían estar causando la infección son e. coli (0.75), estafilococo-coag-pos (0.3), pseudomonas-aeruginosa (0.3), klebsiella-pneumoniae (0.5)
—de la [REGLA 545]

Esta facilidad de explicación es en realidad un recorrido parcial a través del proceso de razonamiento del programa, expresado en la jerga médica. Tales explicaciones se pueden producir de forma bastante rápida y económica en sistemas basados en reglas (devolviendo las reglas que se están empleando y sus predecesoras), lo que constituye un punto a favor de la codificación del conocimiento basada en reglas. Observe que las conclusiones llevan unidas estimaciones numéricas. Éstas en realidad no son probabilidades auténticas. Son estimaciones que permiten que el sistema trate con la incertidumbre de una forma coherente y produzca una lista ordenada de diagnósticos probables en el análisis final.

Un buen consejo...

Se entra al Wheat Counsellor a través del sistema Prestel...

La introducción inicial permite que el usuario seleccione entre "consejo de mediada la temporada" o "planificación otoñal"

El sistema formula entonces al granjero una serie de preguntas sobre cuyas respuestas basará su consejo...

Habiendo interpretado las respuestas del granjero, Wheat Counsellor aconseja en forma de valores de clasificación para los diversos tratamientos químicos disponibles

El cerebro del grano

Wheat Counsellor (Consejero del trigo) es un sistema experto desarrollado por ICI y que los agricultores tienen a su disposición, de forma gratuita, a través del Prestel. El sistema está diseñado para aconsejar a los granjeros acerca de los métodos para combatir las enfermedades del trigo, los productos químicos correctos a aplicar y una estimación de las pérdidas probables. Aunque el sistema está diseñado para simular el tipo de conversación que se podría mantener a través del portal de una granja, su conocimiento se ha acumulado a partir de estudios científicos sobre la difusión de las enfermedades de la cosecha. Con el fin de obtener la información a partir de la cual extraerá sus conclusiones, el sistema conduce al granjero a través de una serie de preguntas sencillas. Luego ofrece su consejo en forma de una lista de los tratamientos disponibles, junto con un porcentaje de conveniencia para cada tratamiento.

Vista ampliada

Examinemos los archivos sobre los que operan las instrucciones del CP/M

Ya hemos hablado sobre las extensiones de archivos en CP/M, pero ahora vamos a estudiarlas con mayor profundidad. Como hemos podido ver, un archivo CP/M consta de un nombre primario, que puede contener hasta ocho caracteres, seguido por un punto y hasta tres letras, que constituyen una ampliación del nombre del archivo (si bien ésta se puede omitir). Teóricamente, al nombre de archivo primario usted puede añadirle cualquier extensión, pero algunas de éstas están reservadas para fines específicos. Por ejemplo, la extensión COM está reservada para archivos de instrucción (COMmand) CP/M. Esta extensión le informa al CP/M que un archivo con dicha extensión se ha de añadir a la lista de instrucciones transitorias que se pueden ejecutar bajo CP/M.

De modo similar, los programas en BASIC escritos bajo CP/M se deben almacenar con la extensión de archivo .BAS. Muchas versiones de BASIC que se ejecutan bajo CP/M suelen asignarle esta extensión de forma automática a un archivo en BASIC, eliminando la necesidad de que el usuario la digite. Aunque el programa en BASIC se almacena como un archivo fuente (texto), la diferencia entre este archivo y un archivo de texto común es que cuando el programa se vuelva a cargar antes de su ejecución, el CP/M distintiva el programa en BASIC en vez de listarlo como un archivo secuencial ASCII. Sin embargo, muchas máquinas compilan sus programas en BASIC antes de ejecutarlos. Los archivos compilados antes de su ejecución se almacenan con una extensión .INT. Esta significa que son archivos INTermediarios que se componen de código objeto.

A los programas en código máquina, como a los programas en BASIC, también se les pueden asignar varias extensiones diferentes según cuál sea su estado. Cuando escribimos un programa en assembly, al listado fuente se le debe añadir la extensión .ASM. Si el programa no posee la extensión correcta, el ensamblador residente no intentará ensamblar el programa y simplemente generará un error de archivo.

Cuando intentamos ensamblar el programa, debemos seleccionar una de dos extensiones. Éstas son .HEX o bien .PRN. La extensión HEX significa que lo que se produzca al final del ensamblaje será un archivo objeto HEXadecimal. Por su parte, el ensamblaje con PRN también producirá un listado impreso del ensamblaje con una copia del listado fuente, el código objeto, las direcciones de cada uno de los opcodes y la lista de asignaciones junto con una lista de errores. Por consiguiente, la extensión PRN ofrece una ayuda esencial para la puesta a punto de programas en assembly.

El último grupo de extensiones de archivos que veremos aquí son aquellas relacionadas con los archivos de texto. Muchas versiones de CP/M poseen una pequeña facilidad para edición de textos (a la que se llama ejecutando el archivo de instrucción ED) que se proporciona con el propio sistema operativo, mientras que otras versiones dan por sentado que usted cuenta con un paquete especializado de tratamiento de textos ejecutable bajo CP/M. No obstante, independientemente de la versión que se esté utilizando, las extensiones de los archivos son las mismas.

Aparte de las extensiones de nombres de archivos reservadas que ya hemos mencionado, se pueden utilizar como extensión tres letras casi cualesquiera. En realidad, no es necesario añadir ninguna extensión en absoluto. Si se guarda (SAVE) un archivo de texto sin haberle añadido ninguna extensión, a menudo el CP/M la añadirá por nosotros. Por ejemplo, la utilidad TEXTED añadirá la extensión .TXT al final de un archivo, donde *WordStar* añadiría .$$$.

Archivos de seguridad

Si, tras haber guardado (SAVE) un archivo de texto, usted observa el directorio, verá que se han creado dos archivos: uno con la extensión de archivo que usted ha añadido y otro con el sufijo .BAK. Ésta es una facilidad de seguridad incorporada en el CP/M. Obviamente, es de importancia vital que los documentos importantes no se puedan borrar o corromper accidentalmente, de modo que para evitar hechos tan desastrosos el CP/M creará dos copias de todos los archivos de texto: un archivo normal y un archivo .BAK de seguridad.

De este modo, si accidentalmente usted borra (ERA) un archivo al editarlo, siempre estará disponible la versión de seguridad, gracias a la cual su trabajo anterior no se habrá perdido de forma irrecuperable. Cuando se ha editado y vuelto a guardar un archivo, también se creará una nueva versión de .BAK. Por lo tanto, es una buena idea volver a guardar (SAVE) regularmente los archivos de modo que la copia de seguridad esté siempre actualizada a la última versión.

A pesar de que no es necesario añadir extensiones, una vez implementadas resultan una valiosa herramienta para organizar sus archivos; no sólo en su propio beneficio (para visualizar información sobre la organización de archivos y encabezamientos), sino también para la manipulación de grupos de archivos. Una de las más útiles de estas manipulaciones supone el uso de caracteres de "máscara", que posibilitan el emparejamiento de archivos.

Caracteres de control del CP/M

El CP/M contiene un cierto número de caracteres de control para llevar a cabo muchas de sus funciones. Aunque varios de ellos están ya obsoletos y se implementan muy raramente, se los conserva en el sistema operativo estándar para preservar la compatibilidad entre diferentes ordenadores

CTRL C	Re-carga el disco de sistema CP/M
CTRL C	Inicializa el disco tras su inserción
CTRL M	Termina las instrucciones PIP y devuelve el control al CP/M
CTRL Z	Devuelve control a ED tras oper. de inser.
CTRL P	Envía las entr. a la impr. Volviendo a digitar CTRL P se cancela esta oper.
CTRL U	Borra la línea en curso
CTRL X	Borra la línea en curso y lleva al cursor de vuelva al principio
CTRL M	Ejecuta la línea de instrucción en curso (utilizado en lugar de RETURN)
CTRL H	Retroceso/borrar
CTRL E	Permite entrar 1 línea de instr. larga sin que se ejecute al llegar al final de la 1.ª l.
CTRL R	Repite la línea de instr. en curso

Supongamos, por ejemplo, que un director gerente ha escrito cierto número de memorándums en el mes de julio. Un tiempo después quizá decida que estos archivos ya están obsoletos y que ya no los necesita en su catálogo. Para suprimir los memorándums de julio puede, por supuesto, recorrer el catálogo y borrar individualmente cada uno de los archivos utilizando la instrucción ERA; pero si el número de archivos es elevado, este proceso será largo y tedioso. Sin embargo, siempre y cuando haya tenido la previsión de darles a todos los memorándums del mes de julio la misma extensión de archivo, pongamos por caso. JUL, este proceso se podrá llevar a cabo con una única instrucción: ERA D:*.JUL (donde D es el nombre de unidad opcional). La colocación del asterisco antes del punto indica al CP/M que ignore el nombre de archivo primario y borre (ERAse) los archivos que lleven la extensión .JUL. Este asterisco también se puede utilizar a la derecha del punto. Empleando asteriscos a ambos lados del punto, la instrucción borrará todos los archivos de un disco, como ERA *.*.

Supresión selectiva

Vamos a suponer que el director gerente conserva todos sus archivos de julio con el sufijo .JUL. Ahora complicaremos el asunto suponiendo, asimismo, que hay una cantidad de documentos importantes que él no desea borrar. En este caso, no cabe lugar al empleo del prefijo*, dado que éste borraría los archivos importantes de julio junto con los memorándums innecesarios. No obstante, nuestro director gerente ha diferenciado sus memorándums de otros archivos otorgándoles los nombres MEM1.JUL,MEM2.JUL, y así sucesivamente. De esta forma todavía es posible borrar todos los memorándums con una sola instrucción sin borrar accidentalmente otros archivos, mediante el uso de la instrucción ERA D:MEM?.JUL. Aquí se está indicando al sistema operativo que borre todos los archivos que comienzan por MEM y terminan con el sufijo .JUL. El ? del cuarto lugar le indica al CP/M que el cuarto carácter no es significativo y que lo puede ignorar.

Este proceso se puede adaptar a cualquiera de las once posibles posiciones en un nombre de archivo. Es particularmente útil en el caso de que nuestro director gerente desee borrar los memorándums tanto de JULio como de JUNio, dado que ambos se reemplazarían por el formato MEM?.JU?. En un nombre de archivo pueden aparecer combinaciones de * y ?, de modo que ERA D:????Q???.* borraría todos los archivos que tuvieran una Q en la quinta posición del prefijo.

El empleo de los caracteres de "máscara" * y ? no está limitado a la instrucción ERA, sino que también se pueden usar con las instrucciones PIP, STAT y REN, entre otras. De modo que, para transferir un número de archivos de un disco a otro, una instrucción típica es PIP B:=A*.*. Esta instrucción buscará todos los archivos de la unidad A que concuerden con el formato (que, en este caso, serían todos ellos) y los copiará en el disco de la unidad B. De esta manera podemos utilizar caracteres de "máscara" para copiar el contenido completo de un disco en otro.

Hasta este momento, a lo largo de la serie hemos hablado de instrucciones que se emplean para controlar el CP/M. Sin embargo, el sistema operativo también hace un uso exhaustivo de caracteres de control para llevar a cabo muchas de sus operaciones. Quizás el carácter de control más útil y más común sea la combinación CTRL-C. La pulsación de estos caracteres producirá una "carga en caliente" del sistema y volverá a cargar el CP/M en el ordenador. Esto es importante, no sólo para volver a cargar el CP/M después de que el sistema quede colgado, sino también al insertar un disco nuevo en una de las unidades.

Recuerde que el CP/M conserva una copia del "enganche" (log) de un disco en la memoria del ordenador. Cuando intercambiamos discos en una unidad, debe informarse del cambio al sistema operativo, de lo contrario podría generar errores. Cuando hacemos una carga en caliente del CP/M, éste reenganchará el contenido de cada unidad de disco, lo que nos permitirá continuar. CTRL-C también se puede utilizar para interrumpir la ejecución de muchos programas, dado que el CP/M interrumpirá automáticamente el sistema al reinicializarse a sí mismo.

Muchos de los otros caracteres de control están relacionados con la edición de textos y los han adoptado numerosos procesadores de textos que operan bajo CP/M. Por ejemplo, CRTL-H borrará el último carácter, mientras que CTRL-U y CTRL-X borrarán una línea entera. Si se conecta una impresora, se puede enviar texto a la misma digitando CTRL-P. La pulsación por segunda vez de CTRL-P interrumpe esta operación.

El CP/M utiliza otros caracteres de control, muchos de los cuales han quedado obsoletos a raíz de los avances en materia de hardware y software. Existen varias razones por las cuales se han conservado estos caracteres, aun cuando sus funciones parecen haberse duplicado mediante otras teclas; por ejemplo, ahora la mayoría de los ordenadores personales llevan instalada una tecla Delete. Una parte de la explicación es de carácter histórico. Muchos de los primeros microordenadores no tenían tecla Delete o teclas para el cursor, de modo que las facilidades para edición en pantalla debían de estar incorporadas en el sistema operativo.

Otro motivo para conservar el sistema es la compatibilidad. Mientras que, con el correr de los años, los caracteres CTRL y alfabéticos han conservado sus equivalentes ASCII, puede que las teclas más nuevas no los tengan; por ello, para preservar la compatibilidad, se han conservado los caracteres de control. Por último, un usuario que se haya pasado varios años con el CP/M se habrá acostumbrado a usar estas teclas. La alteración del sistema supondría que el usuario debería ahora volver a aprender el control desde el principio.

Habiendo terminado nuestro breve resumen de las instrucciones y archivos tal como están implementados en CP/M, estamos ya en condiciones de llevar a cabo la mayor parte de las funciones cotidianas utilizadas en el sistema operativo.

Usando máscaras
El uso de caracteres de "máscara" es una importante característica del CP/M. Entrando caracteres de "máscara" en posiciones diferentes el ordenador, en efecto, ignorará los caracteres que haya en esa posición cuando busque en el directorio. Ello significa que mediante la cuidadosa asignación de nombres de archivo, el usuario puede manipular listas enteras de archivos clasificados bajo encabezamientos diferentes

Liz Dixon

Garabatos digitales

Finalmente, ofrecemos la segunda parte del programa del trazador para el BBC Micro

La segunda parte del programa del trazador controla el uso de éste en una "modalidad elástica". En lugar de simplemente ofrecer una función de dibujo a mano alzada, cuando se selecciona la modalidad elástica se puede utilizar el trazador para especificar los puntos de origen y final de una línea, el centro y el radio de un círculo, o los tres vértices de un triángulo "sólido". La selección de cada una de estas funciones, junto con una función de trazado de puntos, se realiza pulsando las teclas apropiadas a través del menú visualizado en la parte superior de la pantalla. De este menú se pueden seleccionar, asimismo, las funciones Save, Load y Clear (limpiar la pantalla), similares a aquellas para la modalidad a mano alzada mencionadas en el capítulo anterior. Por consiguiente, esta nueva sección permite crear patrones complejos de una forma muy simple, utilizando el trazador y algunas pulsaciones de teclas. Estos patrones se pueden guardar en disco o cinta para volver a cargarlos de nuevo.

Los 4 procedimientos

- PROCpunto

La llamada a PROCdibujar (procedimiento ofrecido en la primera mitad del programa) con flagart establecida en uno, nos proporciona una forma sencilla de mover el cursor por la pantalla sin dibujar líneas. El establecimiento de flagart en uno asegura la selección de la modalidad lápiz arriba antes de que se llame al procedimiento. Esta llamada está incluida dentro de un bucle que también busca una pulsación de tecla que podría terminar el bucle o utilizarse para cambiar el color de primer plano. Además, si se pulsa el botón del trazador, se dibuja un punto en la posición actual del cursor y en el color actual del primer plano. Para evitar pulsaciones dobles, el procedimiento no puede continuar tras haberse detectado una pulsación del botón hasta que se libere el mismo.

- PROClinea

La primera acción del procedimiento es almacenar las coordenadas actuales del cursor, puesto que éstas se emplearán para ubicar el extremo fijo de la línea elástica. Luego se borra el cursor y se traza una línea hasta un punto que corresponde a una nueva posición del cursor. Esta se traza en modalidad de trazado Exclusive-OR, utilizando GCOL3. Si posteriormente se volviera a dibujar esta línea, se borraría y todos los datos del fondo se restablecerían a su condición original. La colocación de estas dos acciones de dibujo dentro de un bucle permite al usuario probar varias posiciones de línea, como

Instrucciones del trazador

Menú principal

Seleccionar	Función
1	Modalidad de mano alzada
2	Modalidad elástica

Menú de mano alzada

Seleccionar	Función
S	Guardar pant. en cinta o disco
L	Cargar pant. de cinta o disco
M	Retornar al menú principal
C	Limpiar pantalla
Botón	Lápiz arriba/lápiz abajo

Menú elástico

Seleccionar	Función
S,M,L,C	Igual que en mano alzada
P	Trazar un punto en el color de primer plano actual. M. retorna al menú elástico
D	Dibujar una línea en el color actual de primer plano desde la pos. del cursor al entrar hasta la pos. actual del cursor. Pulsar M o el botón para fijar la línea y retornar al menú elást.
F	Rellenar un triángulo entre 3 puntos especificados mediante puls. del botón
R	Dibujar un círculo en el color de primer plano actual. El centro está dado por la pos. del cursor a la entrada y el radio varía según la pos. actual del cursor. Pulsar M o el botón para dibujar el círculo y retornar al menú elástico

Colores de primer plano

Seleccionar	Función
1	Selecciona el color lógico 1 como color de primer plano (por defecto: rojo)
2	Selecciona el color lógico 2 como color de primer plano (por defecto: amarillo)
3	Selecciona el color lógico 3 como color de primer plano (por defecto: blanco)

Nota: Los colores lógicos se pueden cambiar utilizando VDU 19 para elegir entre los 16 colores disponibles. P. ej., VDU 19,1,6,0,0,0 cambia el color lógico 1 por cyan (color real 6). El color de primer plano se puede volver a seleccionar en cualquier momento

si la línea fuera una banda elástica fijada por un extremo a un punto fijo y por el otro al cursor de pantalla del trazador. Una vez satisfecho con la posición de una línea, el usuario puede "fijar" la línea pulsando el botón del trazador o bien M en el teclado. En este caso el procedimiento omite dibujar la línea una segunda vez, dejándola sin borrar, antes de retornar al menú principal.

- PROCrellenar

El procedimiento de relleno saca partido de la facilidad de relleno de triángulos del BBC, a la que se accede mediante el uso de una instrucción PLOT 85,x,y. Esta instrucción toma los dos últimos puntos visitados y el punto especificado en la instrucción como los tres vértices de un triángulo y rellena la forma con un bloque sólido de color. El procedimiento se vale de un método similar al empleado en PROCpunto para permitir que el usuario trace los tres vértices del triángulo a rellenar, utilizando el botón del trazador. Las dos últimas líneas del procedimiento vuelven a visitar los tres puntos para rellenar la forma triangular.

- PROCcirculo

La última facilidad que ofrece el programa permite que el usuario especifique el radio de un círculo utilizando el mismo método de banda elástica que el empleado en PROClinea. De hecho, precisamente para este fin se llama a PROClinea desde el procedimiento para dibujar el círculo. Una vez seleccionado el radio, el control retorna a PROCcirculo, que antes que nada debe borrar la línea dejada por PROClinea y borrar el cursor antes de dibujar el círculo. El centro del círculo se retendrá en x1 y x2, las coordenadas del cursor al entrar en el procedimiento, y PROClinea proporciona las coordenadas de un punto de la circunferencia, (x2,y2). Utilizando estos dos puntos se puede calcular el radio del círculo y emplear un algoritmo estándar para dibujarlo.

Dibujando sus propias conclusiones
Añadiendo la segunda parte del programa *Trazador digital*, que ofrecemos aquí, a la sección del capítulo anterior, se puede utilizar el trazador para dibujar líneas rectas, círculos y triángulos "sólidos" para crear visualizaciones en pantalla como éstas

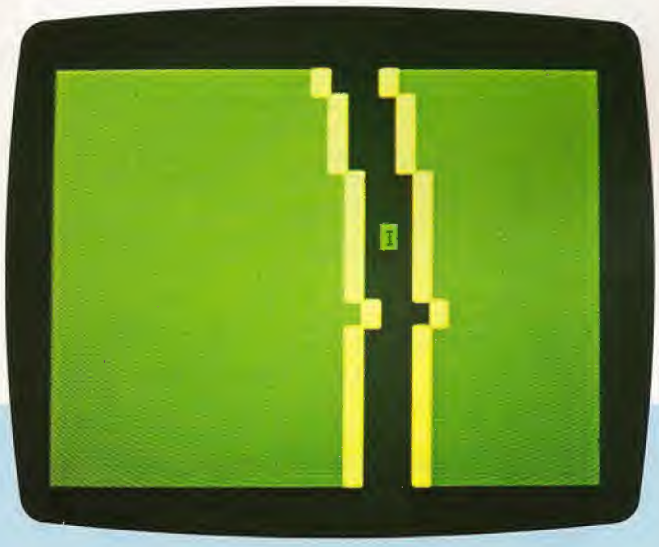
Prog. Trazador digital (II)

El siguiente listado constituye la segunda parte del programa para el trazador digital para el BBC Micro. Se la debe añadir a la primera parte del listado, ofrecida en el capítulo anterior

```
2350 DEF PROCelastica
2360 PROCinforme__elastica
2370 REPEAT
2380 resp$=INKEY$(1):IF resp$<>"" THEN PROCpulsacion__elastica
2390 flagart=1:PROCdibujar
2400 UNTIL flagsalida=1
2410 ENDPROC
2430 DEF PROCinforme__elastica
2440 PROCcalc__xy:antx=x:anty=y:PROCcursor(antx,anty)
2450 PROCreinformar
2460 GCOL 0,1:MOVE 0,920:DRAW 1280,920
2470 ENDPROC
2480 DEF PROCreinformar
2490 PRINT TAB(1,1);SPC(79)
2500 PRINT TAB(1,1);"S=SAVE L=Load M=Menu C=
     Limpiar R=Circulo"
2510 PRINT" D=Dibujar linea   P=Punto   F=Rellenar"
2520 ENDPROC
2540 DEF PROCpulsacion__elastica
2550 IF resp$="C" THEN CLS:PROCinforme__elastica:ENDPROC
2560 IF resp$="M" THEN flagsalida=1
2570 PROCcambiar__color
2580 IF resp$="R" THEN PROCtitulo__circulo:PROCcirculo:PROCinforme__
     elastica:ENDPROC
L.2590,2600
2590 IF resp$="D" THEN PROCtitulo__linea:PROClinea
2600 IF resp$="P" THEN PROCtitulo__punto:PROCpunto:PROCinforme__
     elastica:ENDPROC
2620 IF resp$="S" THEN PROCguardar__pantalla:PROCinforme__
     elastica:ENDPROC
2630 IF resp$="L" THEN PROCcargar__pantalla:PROCinforme__
     elastica:ENDPROC
2640 ENDPROC
2660 DEF PROCtitulo__punto
2670 PRINT TAB(1,1) SPC(79)
2680 PRINT TAB(15,1);"Modalidad punto"
2690 ENDPROC
2710 DEF PROCpunto
2720 REPEAT
2730 flagart=1:PROCdibujar
2740 resp$=INKEY$(1)
2750 PROCcambiar__color
2760 IF (ADVAL(0)AND 3)<>0 THEN PLOT 69,x,y:REPEAT UNTIL
     (ADVAL(0) AND 3)=0
2770 UNTIL resp$="M"
2780 PROCreinformar
2790 ENDPROC
2800 DEF PROCtitulo__linea
2810 PRINT TAB(1,1) SPC(79)
2820 PRINT TAB(15,1);"Modalidad linea"
2830 ENDPROC
2850 DEF PROClinea
2860 X1=x:y1=y:REPEAT
2880 PROCcalc__xy:PROCcursor(antx,anty)
2890 x2=x:y2=y:GCOL 3,colour:MOVE x1,y1:DRAW x2,y2
2900 resp$=INKEY$(1)
2910 IF (ADVAL(0) AND 3)<>0 THEN resp$="M"
2920 PROCcursor(x,y):antx=x:anty=y
2930 IF resp$<>"M" THEN GCOL3,colour:MOVEx1,y1:DRAW x2,y2
2940 PROCcambiar__color
2950 UNTIL resp$="M"
2960 PROCreinformar
2970 ENDPROC
2990 DEF PROCtitulo__rellenar
3000 PRINT TAB(1,1) SPC(79)
3010 PRINT TAB(15,1);"Modalidad rellenar"
3020 ENDPROC
3040 DEF PROCrellenar
3050 FOR I=1 TO 3
3060 REPEAT
3070 flagart=1:PROCdibujar:resp$=INKEY$(1)
3075 IF resp$<> "" THEN PROCcambiar__color
3080 UNTIL (ADVAL(0)AND 3)<>0
3090 PLOT 69,x,y:x(I)=x:y(I)=y
3110 REPEAT UNTIL (ADVAL(0)AND 3)=0:REM ESPERAR SOLTAR
     BOTON
3120 NEXT I
3155 PROCcursor(x,y):REM cursor apagado
3140 FOR I=1 TO 2:PLOT 69,x(I),y(I):NEXT I
3150 PLOT 85,x,y
3160 ENDPROC
3180 DEF PROCtitulo__circulo
3190 PRINT TAB(1,1) SPC(79)
3200 PRINT TAB(15,1);"MODALIDAD CIRCULO"
3210 ENDPROC
3230 DEF PROCcirculo
3240 PROClinea
3250 GCOL 3,colour:MOVE x1,y1:DRAW x2,y2:REM BORRAR
     LINEA
3260 GCOL 0,colour:REM VOLVER A MODALIDAD TRAZADO
     NORMAL
3270 PROCcursor(x2,y2)
3370 radio=SRQ((x2-x1)^2+(y2-y1)^2)
3380 MOVE x1+radio,y1
3390 FOR angulo=0 TO 2*PI STEP 0.1
3400 rx=x1+radio*COS(angulo)
3410 ry=y1+radio*SIN(angulo)
3420 DRAW rx,ry
3430 NEXT angulo
3435 DRAW x1+radio,y1
3440 ENDPROC
```

Autopista

He aquí un ejercicio peligroso, que desaconsejamos a quienes sean poco hábiles. Está escrito en BASIC para el micro Alice, de Matra. No olvide releer las líneas del listado



Usted debe atravesar las cuatro vías de la autopista a una hora en que la circulación es extremadamente densa. Afortunadamente un terraplén central le permite descansar un poco antes de iniciar la segunda parte de la travesía. Cada travesía finalizada con éxito vale un punto. Si usted se estrella, pierde una de sus cinco vidas y el juego se reanuda después de un pequeño intermedio musical. Pulse las teclas W para avanzar y Z para retroceder. Un buen consejo: mire bien a ambos lados antes de entrar en la calzada.

```
5 REM ********************
10 REM *      AUTOPISTA      *
12 REM ********************
13 REM
14 REM INICIALIZACION
15 REM
19 REM RESERVA ESPACIO CADENAS
20 CLEAR 250
30 CLS 0
40 A$=" "
50 B$=" "
60 S=0
70 NP=5
80 P=367
90 P1=P
100 P$=CHR$(191)
110 RESTORE
115 REM INICIALIZACION COCHES
120 FOR I=1 TO 32
130 READ A,B
140 A$=A$+CHR$(A)
150 B$=B$+CHR$(B)
160 NEXT I
170 PRINT @ 64
180 PRINT @ 96
190 PRINT @ 192
200 PRINT @ 288
210 PRINT @ 320
220 PRINT @ P,P$
230 X=RND(31)
240 A$=RIGHT$(A$,X)+LEFT$(A$,32-X)
244 REM
245 REM BUCLE PRINCIPAL
246 REM
250 PRINT @ 352, "VIDA(S) REST.";NP;
255 REM DESPLAZAMIENTO COCHES
260 A$=RIGHT(A$,1)+LEFT$(A$,31)
270 B$=RIGHT(B$,31)+LEFT$(A$,1)
280 PRINT @ 128,B$;
290 PRINT @ 256,A$;
300 PRINT @ 160,A$;
310 PRINT @ 224,B$;
315 REM MOVIMIENTO JUGADOR
320 D$=INKEY$
330 P=P+32*((D$="Z")-(D$="W"))
340 IF P>367 THEN P=367
345 REM RUTA ATRAVESADA?
350 IF P=111 THEN 450
360 C=PEEK(16384+P)
365 REM PERDU?
370 IF C<>96 AND C<>128 AND C<>191
    THEN 600
380 PRINT @ P1," ";
390 PRINT @ P,P$;
400 P1=P
410 D$=INKEY$
420 IF D$<>" " THEN 330
430 GOTO 250
444 REM
445 REM RUTA ATRAVESADA
446 REM
450 PRINT @ P1, " ";
460 PRINT @ P,P$;
470 SOUND 1,1
480 FOR I.0041 TO 200
490 NEXT I
500 PRINT @ P," ";
510 P=367
520 P1=P
530 S=S+1
540 PRINT @ 0, "PUNTOS :";S,
550 PRINT "RECORD :";R
560 GOTO 230
594 REM
595 REM PERDIDO
596 REM
600 NP=NP-1
610 PRINT @ P,CHR$(191);
620 PRINT @ P1," ";
630 GOSUB 1000
640 IF NP=0 THEN 700
650 P=367
660 P1=P
670 GOTO 230
700 IF S>R THEN R=S
710 CLS
720 PRINT   256,"PUNTOS :";S,
730 PRINT "RECORD :";R;
740 PRINT @ 330,"OTRA?";
750 D$=INKEY$
760 IF D$=" " THEN 750
770 IF D$<>"N" THEN 30
780 END
994 REM
995 REM MARCHA FUNEBRE
996 REM
1000 SOUND 15,12
1010 SOUND 15,9
1020 SOUND 15,3
1030 SOUND 15,12
1040 SOUND 55,9
1050 SOUND 45,3
1060 SOUND 45,9
1070 SOUND 15,3
1080 SOUND 15,9
1090 SOUND 1,3
1110 D$=INKEY$
1120 RETURN
2000 DATA 128, 128, 151, 151, 155, 155, 128, 128, 128, 128,
     128, 151, 128, 159
2010 DATA 151, 128, 155, 128, 128, 128, 128, 128, 128, 128,
     151, 128, 155
2020 DATA 128, 128, 128, 128, 147, 151, 155, 155, 128, 128,
     128, 151, 128, 155
2030 DATA 151, 128, 155, 128, 128, 151, 159, 159, 128, 128,
     128, 128, 128
2040 DATA 128, 128, 151, 128, 155, 151, 128, 155
```

El esperado regreso de Atari

La firma Atari hace su reaparición en el mercado con el microordenador 130XE, que se destaca por su elegante diseño y sus 128 Kbytes de memoria

Chris Stevens

Una nueva imagen
El Atari 130XE es el primer producto que la empresa lanza bajo la dirección del actual presidente, Jack Tramiel. Con 128 K de RAM a bordo y a un precio que sale muy favorecido frente al de muchos ordenadores con sólo la mitad de memoria, Atari confía en que sea ésta la máquina que ayude a la empresa a recuperar su fortuna

A pesar del hecho de que Atari fuera uno de los pioneros en el mercado del ordenador personal, la primera parte de los años ochenta no fue buena para esta firma. En 1984, tras sufrir severas pérdidas, se hizo cargo de Atari el ex director de Commodore, Jack Tramiel, quien se abocó a la tarea de invertir la suerte del achacoso gigante de los ordenadores. Su política de comercializar tecnología de ordenadores al menor precio posible se hizo evidente enseguida en las tiendas, cuando los precios de los micros 600XL y 800XL descendieron sustancialmente de cara al mercado navideño.

Esta jugada no fue suficiente para invertir la tendencia. Atari se enfrentaba a un círculo vicioso de la comercialización de ordenadores: la disminución de las ventas produce falta de apoyo de software, lo que a su vez contribuye a reducir las ventas aún más, lo que significa escasez de fondos para invertir en máquinas nuevas.

No deja de ser irónico el hecho de que fuera Jack Tramiel la persona que más responsabilidad tuvo en colocar a Atari en esa crítica posición. Su agresiva comercialización del Commodore 64 hizo que Atari fuera casi barrida por su rival. A comienzos de 1985 la situación estaba cambiando radicalmente. Commodore se encontraba con una caída de sus ventas y el fracaso del Plus-4 en producir un gran impacto en el mercado, mientras Atari anunciaba un sinnúmero de nuevos productos. La primera de las nuevas máquinas es el Atari 130XE, un ordenador basado en el procesador 6502C.

El 130XE es esencialmente la misma máquina que el ordenador Atari de ocho bits básico, que, en una u otra forma, ha estado en el mercado desde principios de los ochenta. La principal diferencia entre esta máquina y los primeros ordenadores Atari es el elegante nuevo estilo y la cantidad masiva de memoria: el Atari 130XE posee 128 K completos de RAM.

La carcasa del ordenador tiene un aire muy distinto al de sus predecesores. La armazón externa, ligera y de plástico gris, posee el elegante diseño que el público espera de un ordenador moderno, con líneas redondeadas y teclas anchas y esculpidas que facilitan la digitación. Las teclas poseen un recorrido ligeramente mejor que el que ofrecían los modelos anteriores, con la ventaja adicional de no traquetear cuando uno escribe.

Al igual que los otros micros Atari basados en el 6502, el 130XE posee cinco teclas de función preprogramada. Sin embargo, a diferencia de los modelos previos, éstas se han colocado encima del teclado principal en vez de sobre el lado derecho. Ahora están moldeadas en el mismo plástico gris que el resto de la carcasa y están diseñadas en una elegante forma de paralelogramo y no de cuadrados. Éstas representan una gran mejora respecto a las de la serie XL, que eran de metal y su tacto era claramente inestable e inseguro.

Conexiones de interface

Las interfaces instaladas atrás y a la derecha del nuevo Atari también deparan pocas sorpresas. En el costado están las esperadas puertas para palanca de mando tipo D de nueve patillas utilizadas por Atari y adoptadas desde entonces para casi todas las máquinas. En la parte posterior de la máquina está la puerta de control en serie de 13 patillas que emplea Atari para acoplar y enlazar en margarita periféricos tales como aparatos de cassette, unidades de disco e impresoras. A la derecha, insertadas en la carcasa, están las interfaces para cartucho y de ampliación.

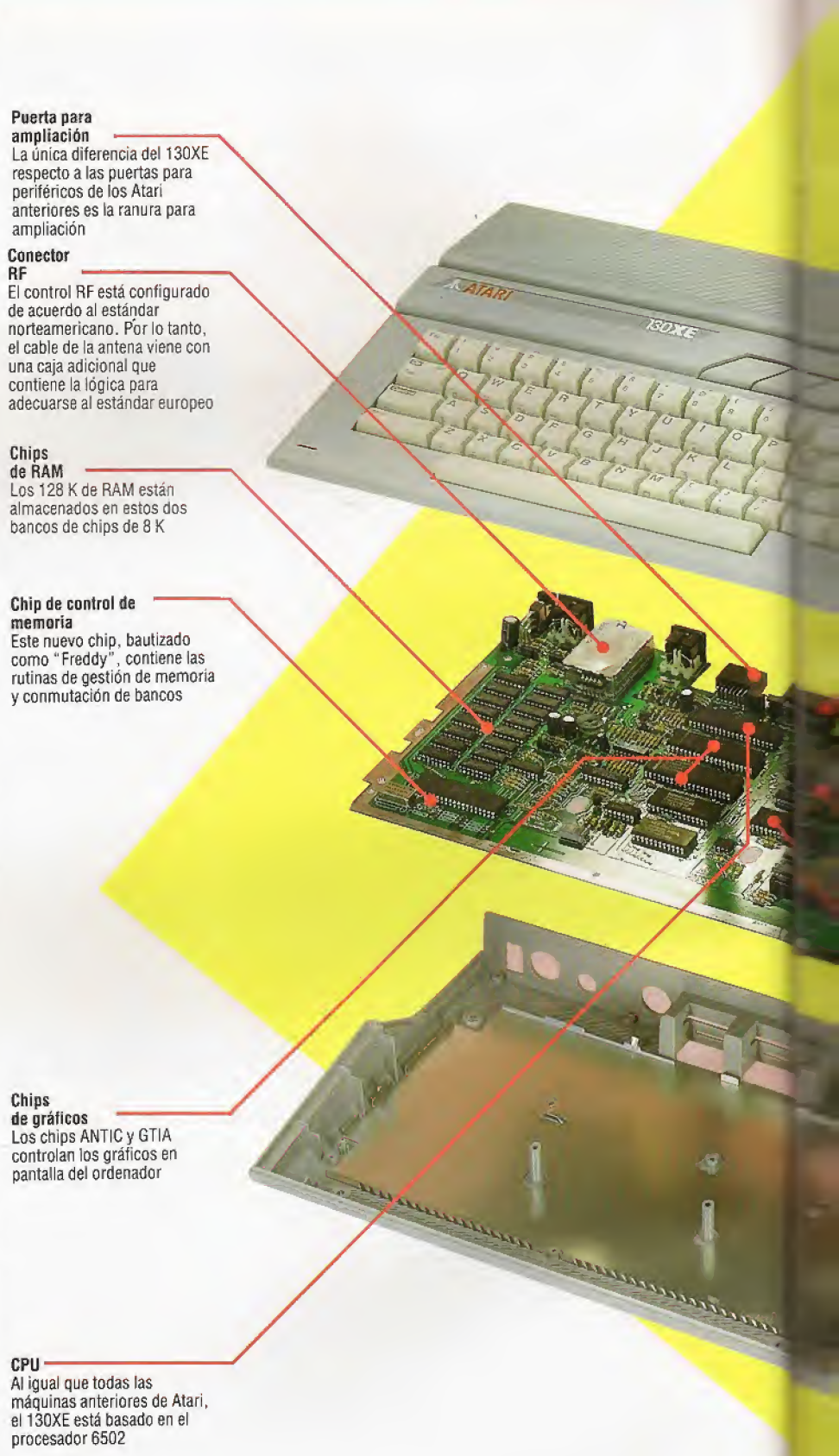
La puerta para cartuchos permite a la máquina ejecutar la vasta cantidad de cartuchos de juegos Atarisoft, como *Pacman* y *Galaxians*, que con el correr de los años han sido uno de los puntos fuertes de la empresa.

La puerta para ampliación, sin embargo, se aparta un tanto del estándar anterior. Los primeros ordenadores Atari tenían como bus de ampliación un conector marginal de 50 vías. La máquina nueva tiene instalada como conector de cartuchos un bus mucho más pequeño, de 14 vías. Las interfaces restantes del 130XE son el conector para monitor compuesto, un enchufe hembra RF para aparatos de televisión y el conector para fuente de alimentación Atari estándar.

Una gran memoria

La elegancia del estilo y la compatibilidad están muy bien, por supuesto, pero Atari también las había aplicado a la serie XL sin ningún éxito llamativo. Lo que distingue al 130XL de las primeras máquinas y lo que constituye su evidente punto fuerte de ventas, es la disponibilidad de una gran cantidad de memoria a un precio reducido. Un microprocesador de ocho bits, por supuesto, puede direccionar sólo 64 K de RAM a la vez. Para direccionar el doble de esa cantidad, el ordenador ha de hacer uso del proceso que se conoce como *conmutación de bancos*. Utilizando esta técnica, el ordenador puede "mirar" una ventana de 64 K de los 128 K totales. Si bien la técnica no es perfecta, y aunque las instrucciones extras necesarias para conmutar de un banco de RAM a otro conducen a unas velocidades de recuperación algo más lentas, sí permite que un micro de ocho bits direccione más memoria de la que, en otro caso, sería posible.

En la actualidad la técnica de la conmutación de bancos es más común de lo que cabría pensar. Tanto el Oric Atmos como el Commodore 64 incorporan más de 64 K, y ambos utilizan la conmutación de bancos como un medio para hacer un uso cabal de la memoria disponible.



Lo que es inusual en el 130XE es su precio. Aunque más caro que el Sinclair Spectrum y el Acorn Electron, el 130XE es considerablemente más económico que los precios de lista del BBC Micro y el Commodore 64. Para poder proporcionar un ordenador de 128 K a ese precio y obtener beneficios, Atari necesita recortar de forma considerable los costos de producción. Hasta cierto punto, la naturaleza de la máquina en sí misma ha mantenido reducido el precio: el 130XE es, esencialmente, una máquina remendada, lo que significa que los costos de desarrollo se han mantenido en un mínimo. Las economías más importantes han tenido lugar en el interior de la máquina.

Recortando costos

El área de memoria comprende 16 chips de RAM de 8 K. El costo de producción de estos chips, que ya no se consideran como un producto de la "tecnología de vanguardia", ha caído dramáticamente en los últimos años y esto se ha reflejado en el precio. Otra forma de reducir los costos es mantener en un mínimo la cantidad de componentes de la placa. Aunque muchos de los chips de la anterior serie XL se han incluido en el 130XE con el fin de preservar la compatibilidad, el trazado de la placa de circuito impreso es excepcionalmente bueno, y ofrece un aspecto mucho menos abarrotado que el de muchas máquinas cuya capacidad de memoria es de apenas la mitad.

Por último, la firma Atari ha hecho fuertes inversiones en plantas de ensamblaje automatizadas, de las cuales el 130XE es el primer producto. Todos los componentes de la placa se sueldan mediante máquinas.

Dado que, en esencia, no se ha alterado ninguno de los chips de ROM, de gráficos y de sonido, para el usuario el ordenador es exactamente el mismo que los modelos anteriores, con los excelentes gráficos y sonido que son propios de Atari. Uno de los cambios fundamentales que redundan en beneficio del usuario es el manual. Los manuales de consulta que acompañaban a los modelos anteriores estaban simplificados hasta el punto de que parecían estar dirigidos a niños. El tutor de BASIC está muy mejorado y la empresa ha ofrecido en el apéndice algunas especificaciones técnicas. No obstante, para una explicación completa sobre el dialecto, continúa siendo necesario adquirir el Manual de Referencia del BASIC Atari.

Aunque la gama de micros Atari reclama una mejora urgente, la llegada del 130XE es en cierto modo un enigma. Los 64 K extras de RAM le proporcionan al programador muchísima más memoria, pero sin embargo aún no hay ningún programa disponible para sacar partido de ellos, aun teniendo en cuenta la compatibilidad con el software Atari ya existente en el mercado. Normalmente, uno esperaría que el lanzamiento de una máquina como ésta fuera una maniobra preliminar a una seria arremetida destinada al mercado de pequeña gestión. Sin embargo, la nueva dirección de Atari ha negado taxativamente que tuvieran esas intenciones. Quizá la verdadera razón del lanzamiento del 130XE es que Atari pretendía anticiparse al lanzamiento del Commodore 128, una máquina compatible con el Commodore 64 que también posee memoria extra pero cuyo precio es más elevado.

ATARI 130XE

DIMENSIONES
350×233×63 mm

CPU
6502C operando a 1.79 MHz

MEMORIA
128 K de RAM, 24 K de ROM

PANTALLA
Visualización de textos de 40×24, 320×192 pixels (alta resolución) con 256 colores disponibles

INTERFACES
Puerta para cartuchos, enchufe TV, conector para monitor compuesto, dos puertas para palanca de mando, puerta de entrada/salida en serie, interface para ampliación

LENGUAJES DISPONIBLES
BASIC Atari, LOGO, FORTH, PILOT

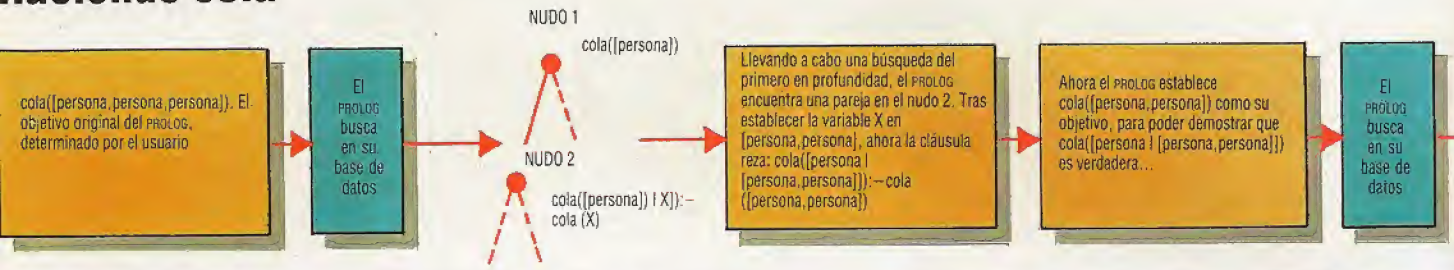
TECLADO
62 teclas, incluyendo cinco teclas de función preprogramadas

DOCUMENTACION
El manual ofrece una explicación completa sobre el BASIC Atari, si bien el tono general sigue siendo de una simplificación excesiva. En el apéndice se ofrecen explicaciones sobre las características de las interfaces y cómo conmutan los otros 64 K de RAM

VENTAJAS
El 130XE posee la ventaja de ofrecer los puntos fuertes ya tradicionales de Atari, y posee 64 K adicionales de RAM que se pueden utilizar como un "disco de silicio" para almacenamiento y recuperación rápidos

DESVENTAJAS
El ordenador no resuelve el problema básico de Atari, el de carecer de una amplia base de software de firmas independientes. El hecho de que el 130XE sea esencialmente una versión remozada de una máquina que lleva ya varios años en el mercado, podría indicar que no generará el interés que tanto necesita la empresa para reafirmar su futuro financiero

Haciendo cola



Búsqueda profunda

A la hora de preguntar
Este diagrama de flujo muestra al PROLOG en acción, respondiendo a una sencilla pregunta formulada por el usuario. Observe que durante la ejecución del programa, a la variable X se le dan dos valores diferentes, manteniendo, no obstante, ambos valores. Esto es posible porque el PROLOG trata a las variables como locales de cada invocación separada de una cláusula

Mientras que lenguajes como el BASIC y el PASCAL poseen flujos de control secuenciales, pasando el control de sentencia en sentencia por estricto orden de arriba abajo (a menos que un bucle o un GOTO lo interrumpa), en PROLOG el flujo de control asume la forma de una búsqueda del primero en profundidad a través de las cláusulas del programa.

Pensemos en el programa como un árbol, con el objetivo (proposición) a demostrar en la raíz y todos los subobjetivos como puntos de elección donde se dividen las ramas inferiores. Existen muchas formas de buscar en un árbol como éste, pero el método que emplea el PROLOG consiste en tomar la rama situada más a la izquierda y seguirla hasta la mayor profundidad posible. Mientras va probando cada rama, marca su recorrido, y cuando llega abajo y no puede seguir adelante, retrocede hasta el punto de elección más próximo, tomando la rama situada más a la izquierda *que aún no haya tomado* y continúa su avance a partir de allí. De esta forma, el sistema explorará todos los caminos a través del árbol y habrá intentado todas las formas posibles de demostrar el objetivo situado arriba de todo.

Una búsqueda del primero en profundidad que sea exhaustiva, ofrece plenas garantías de cubrir todos los caminos, pero puede llegar a ser un asunto muy largo. De hecho, el PROLOG consigue ahorrarse a sí mismo algo de trabajo. Una cláusula del PROLOG es, como ya hemos visto en el capítulo anterior, una regla que dice que el objetivo del encabezamiento es verdadero y los subobjetivos también:

```
objetivo:–subobjetivo1,subobjetivo2,
subobjetivo3... etc.
```

y así sucesivamente. Probablemente esto se comprenda mejor así:

```
IF subobjetivo1 es verdadero
   AND subobjetivo2 es verdadero
   AND subobjetivo3 es verdadero
   AND etc.
THEN objetivo es verdadero.
```

Dado que los objetivos están relacionados entre sí mediante AND, toda la cláusula fracasará si no se puede demostrar alguno de ellos. De modo que el PROLOG va trabajando a través de los subobjetivos de izquierda a derecha y, si no consigue demostrar alguno, abandona en ese punto y ya no se preocupa por el resto.

El retroceso puede hacer que los programas se comporten de una forma que haga que el orden por el cual están escritas las instrucciones sea casi irrelevante. Sin embargo, la ventaja es que el flujo de control, en vez de ser una cuestión fundamental como lo es en BASIC, tiene una importancia sólo menor, dejando que usted se concentre en la estructura lógica de su problema.

El hincapié que hace el PROLOG en una sentencia "declarativa" del problema no significa que usted no pueda ver comportarse a sus programas de forma procesal. La cláusula en PROLOG:

```
marciano(X):–num. de extremidades
   (X,7),num. de cabezas (X,2),
   sabe programar en (X,cobol).
```

se puede leer declarativamente como: "X es un marciano si X posee 7 extremidades, dos cabezas y sabe programar en COBOL". Su lectura procesal sería: "Para demostrar que X es un marciano, demostrar primero que posee siete extremidades, luego, que posee dos cabezas, luego, que sabe programar en COBOL".

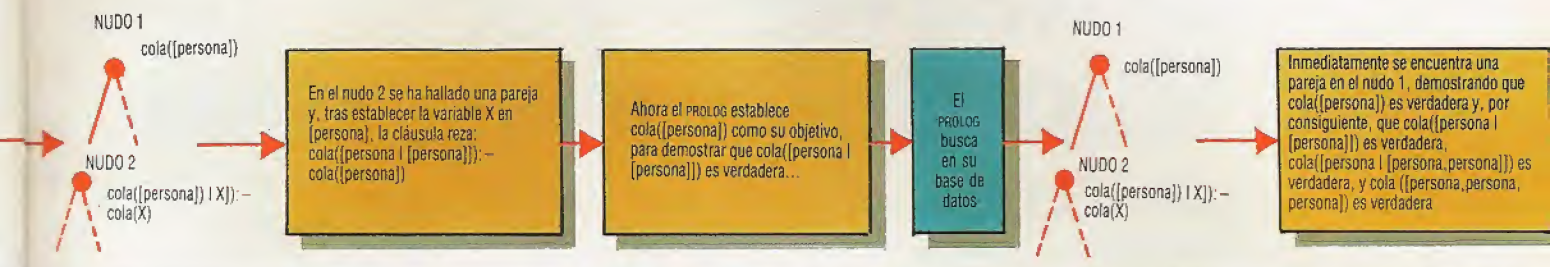
El PROLOG no tiene una "tipología" tan severa como la mayoría de los otros lenguajes. Tampoco es exigente en cuanto a los tipos de datos de los argumentos de sus términos. De modo que el término pred(Argumento) se podría utilizar en ocasiones diferentes con la variable Argumento establecida como entero, como un "átomo" (como marty, venusiano, d24, etc.), o como una lista. No obstante, el PROLOG trata a sus tipos de datos de formas diferentes, permitiendo, por ejemplo, manipular números aritméticamente.

Quien esté familiarizado con el LOGO o el LISP ya se habrá encontrado antes con el tipo de datos lista y las formas especiales que se emplean para manipular listas. En PROLOG una lista se escribe encerrada entre corchetes con los elementos de la lista separados mediante comas. De modo que [manzana,pera,banana] es una lista de frutas, [a,f,e,g,r,x] es una lista de letras y así sucesivamente.

Para penetrar en las listas, el PROLOG nos permite separar un elemento por vez mediante la supresión del primer elemento. La notación [Encabezamiento/Cola] describe una lista con el elemento Encabezamiento y el resto de la lista en la lista Cola. La aplicación de "I" a nuestra lista de frutas nos dará [manzana I [pera,banana]]. Como puede ver, las listas pueden tener como miembros a otras listas. Como caso especial, si tomamos la lista [z80] con un elemento y la descomponemos con I, obtenemos [z80 I []]. La lista de la cola es [], que representa la lista vacía.

El PROLOG permite la recursión que, en realidad, es el estilo normal de un programa en PROLOG. Una definición recursiva es aquella que define algo en términos de sí mismo. En PROLOG podríamos escribir:

```
cola([persona]).
cola([persona I X]):–cola(X).
```



Kevin Jor

Cuando hay dos o más cláusulas, como tenemos aquí, que poseen el mismo encabezamiento, se dice que se trata de un *procedimiento*. El procedimiento cola tiene una cláusula que define a una cola como una lista que posee un elemento, persona. Luego posee una segunda cláusula que nos informa que una cola también podría ser una lista con el elemento persona en el encabezamiento y con una lista llamada X como su cola. Luego, a partir del lado derecho de esta cláusula, vemos que X debe ser una cola. Si le damos al PROLOG un objetivo como:

```
cola([persona,persona,persona]).
```

solicitándole que nos diga si la lista [persona,persona,persona] es una cola. primero busca en su base de datos una cláusula para emparejar con nuestro objetivo. La primera que hallará será cola([persona]). Ésta no concuerda porque las listas no son idénticas, de modo que seguirá explorando hacia abajo hasta la cláusula siguiente: cola([persona | X]):–cola(X). Ésta tampoco concuerda y rellena los valores de las variables de este modo:

```
cola([persona | [persona,persona]]):–
    cola([persona,persona]).
```

Para demostrar que el objetivo del encabezamiento es verdadero, debe demostrar que el subobjetivo lo es. De modo que el PROLOG toma cola([persona,persona]) como su objetivo y comienza a explorar hacia abajo las cláusulas *desde arriba* para hallar una pareja. Nuevamente, cola([persona]) no concuerda, pero la segunda cláusula sí:

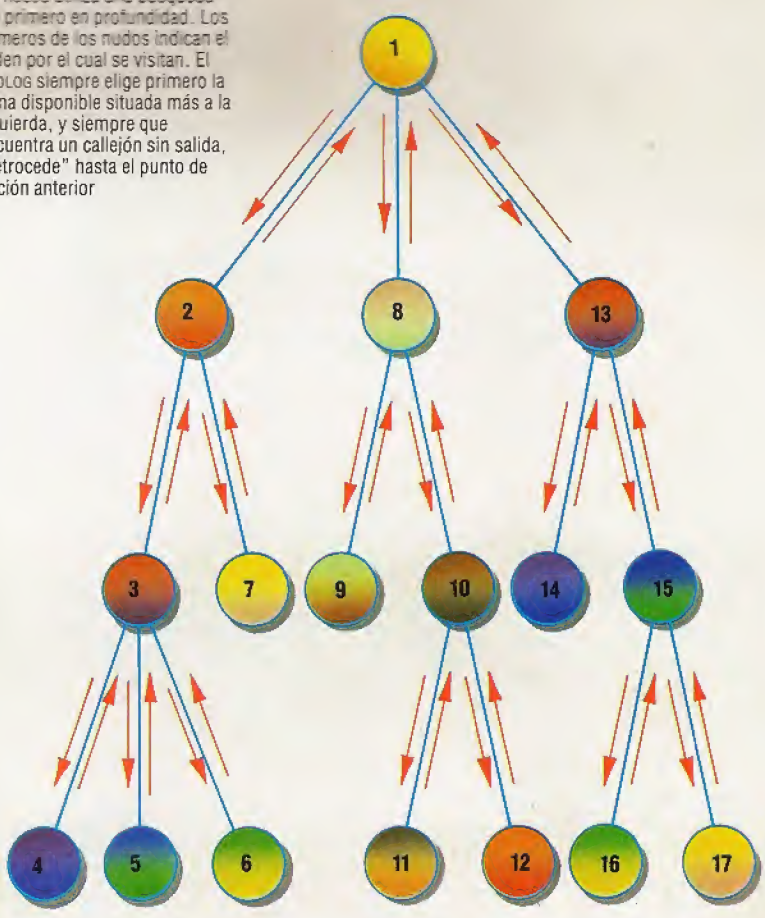
```
cola([persona | [persona]]):–
    cola([persona]).
```

Un punto importante que se debe notar aquí es que la variable X, que en la primera ejecución se estableció en [persona,persona], ahora se ha establecido en [persona]. Pero se ha conservado el primer valor de X. Esto es posible porque las variables son locales a cada invocación separada de una cláusula; por tanto, se puede pensar en cada llamada a cola([persona | X]) como si se empleara una variable única y separada. En PROLOG no existe nada que se parezca a una variable global.

Ya emparejado el encabezamiento de este objetivo, para demostrarlo necesitamos demostrar el subobjetivo a la derecha del símbolo :–, que es cola([persona]). Se inicia otra exploración de la cláusula base y esta vez se encuentra una pareja con cola ([persona]), que encaja directamente y significa que ahora cola([persona, persona]) (nuestro objetivo previo) resulta ser verdadera. Esto a su vez significa que cola([persona,persona, persona]), el objetivo original, es verdadero.

El procedimiento cola nos muestra varias cosas importantes sobre el PROLOG. Por ejemplo, que la disposición de las cláusulas puede ser crucial. (Pruebe colocar las dos cláusulas en el orden contrario y vea lo que sucede al intentar demostrar un objetivo.) Asimismo, ilustra cómo dar cláusulas adicionales es lo mismo que dar formas alternativas de proporcionar un objetivo, tal como si hubiéramos utilizado un OR lógico entre las cláusulas. Ello significa que el PROLOG no necesita de un operador OR, si bien éste se suministra en la mayoría de las implementaciones.

Moviéndose hacia la izquierda
Para explorar su base de datos, el PROLOG utiliza una búsqueda del primero en profundidad. Los números de los nudos indican el orden por el cual se visitan. El PROLOG siempre elige primero la rama disponible situada más a la izquierda, y siempre que encuentra un callejón sin salida, "retrocede" hasta el punto de opción anterior



Kevin Jones

El Nuevo Mundo (I)

Dedicamos los últimos capítulos a considerar los "complementos al BASIC" necesarios

El programa se escribió para el Commodore 64, pero utilizando un BASIC mínimo allí donde ello es posible. Los problemas de conversión para ejecutar el programa en el Spectrum quedan comprendidos en dos áreas principales: primero, el Spectrum sólo permite utilizar nombres de variable de letra única en matrices o contadores de bucles FOR...NEXT. Ofrecemos aquí una tabla de conversión. En segundo lugar, la manipulación de series en el Spectrum es inusual en tanto y en cuanto no se dispone de LEFT$, MID$ ni RIGHT$, si bien todas poseen un equivalente en el Spectrum. Los usuarios del Spectrum habrán de referirse, para estas conversiones, a los complementos que hemos ido ofreciendo con cada módulo. Además, los PRINT CHR$(147) se han de reemplazar por CLS, y las líneas que aguardan pulsaciones de tecla con la forma:

```
<n.º línea>GET I$:IF I$=""THEN<n.º línea>
```

se han de reemplazar por:

```
<n.º línea>LET I$=INKEY$:IF I$="" THEN
GO TO <n.º línea>
```

En el próximo capítulo ofreceremos la segunda parte del listado completo de este programa.

Conversión de variables para el Spectrum

Equivalente Microsoft	Finalidad	Spectrum
TS(,)	Categoría/fortaleza trip.	T(,)
WG()	Tasas salariales	W()
CC()	Contadores categ. de trip.	Y()
PA()	Provisiones adquiridas	A()
PC()	Costo de las provisiones	C()
PN()	Necesidades aprovision.	N()
OC()	Costos merc. a comerciar	O()
OA()	Cant. merc. a comerciar	G()
HR()	Indic. de medias raciones	H()
RR()	Indicadores de eventos	(R)
AO()	Cant. merc. comerciadas	E()
EQ(,)	Valores de intercambio	Q(,)
V1()	Valores al zarpar	B()
V2()	Valores al regresar	D()
S1	Cont. reducción fortalezas	S
S3	Contador impresión lenta	S
S4	Contador breve demora	J
S5	Contador larga demora	S

```
1 REM ***************
2 REM **      Juego      **
3 REM **    Mercantil    **
4 REM ** Nuevo Mundo **
5 REM ***************
6 :
```

La primera sección del programa inicializa las variables y matrices que se requerirán después

```
9 K$="   PULSE CUALQUIER TECLA PARA CONTINUAR"
10 DIM TS(16,2):REM TIPO/FORTALEZA DE LA TRIPULACION
11 CN=0:REM NUMERO DE TRIPULANTES
12 MO=2000: REM DINERO INICIAL
13 DIM WG(5):WG(1)=10:WG(2)=25:WG(3)=15:WG(4)=20
   :WG(5)=15:REM SALARIOS
14 WT=0:REM FACTURA SALARIAL SEMANAL
15 CM=16:REM TRIPULACION MAX
16 DIM C$(5):C$(1)="MARINERO":C$(2)="MEDICO":
   C$(3)="MECANICO"
17 C$(4)="OFICIAL":C$(5)="COCINERO"
18 DIM CC(5):REM CONTADOR DE CADA TIPO DE TRIPULACION
19 REM **** MATRICES DE APROVISIONAMIENTO ****
20 DIM U$(4):U$(1)="KILO":U$(2)="KILO":U$(3)="KILO":
   U$(4)$="BARRIL"
21 DIM P$(4):P$(1)="VEG":P$(2)="FRUTA":P$(3)="CARNE":
   P$(4)="AGUA"
22 DIM PA(4)
23 DIM PC(4):PC(1)=.5:PC(2)=1:PC(3)=2:PC(4)=.5
24 DIM PN(4): PN(1)=2:PN(2)=1:PN(3)=1:PN(4)=.5
30 DIM OA(6)
31 DIM D$(6)
32 D$(1)="FRASCO DE MEDICINA":D$(2)="ARMA":D$(3)="BOLSA DE
   SAL"
33 D$(4)="BALA DE TELA":D$(5)="CUCHILLO":D$(6)="JOYA"
34 DIM OC(6)
35 OC(1)=1:OC(2)=5:OC(3)=.2
36 OC(4)=2:OC(5)=.5:OC(6)=1
40 JL=8:REM DURACION DEL VIAJE
41 EW=0: REM SEMANAS EXTRAS
42 DIM RR(16)
43 REM INDICADORES PARA MOSTRAR SI YA HA OCURRIDO EL EVENTO
   AZAR(N)
44 RC=0
45 REM CONTADOR EVENTO AL AZAR HASTA AHORA
46 RM=13
47 G$="N":REM INDICADOR DE BUEN TIEMPO PARA USAR CON FACTOR
   MOTIN
48 A$="N":B$="N"
49 DIM M(6):REM SEÑALA SI YA SE HAN PRODUCIDO LOS EVENTOS
   MAYORES
60 DIM TS(3):T$(1)="PERLAS":T$(2)="FIGURILLAS":T$(3)
   ="ESPECIAS"
61 DIM V1(3):V1(1)=2:V1(2)=2:V1(3)=1
62 DIM V2(3):V2(1)=2+(INT(RND(1)*1)/2):V2(2)=2+(INT(RND(1)*
   3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIM EQ(4,3)
65 EQ(1,1)=0.5:EQ(1,2)=0.5:EQ(1,3)=1
66 EQ(2,1)=5:EQ(2,2)=5:EQ(2,3)=10
67 EQ(3,1)=3:EQ(3,2)=3:EQ(3,3)=6
68 EQ(4,1)=2:EQ(4,2)=2:EQ(4,3)=4
69 DIM AO(3)
80 PRINT CHR$(147):S$="      JUEGO MERCANTIL NUEVO
   MUNDO*":GOSUB 9100:PRINT
81 GOSUB 9200
82 S$="ERES EL CAPITAN DE UN BARCO*":GOSUB 9100:PRINT
83 S$="QUE SE DIRIGE AL NUEVO MUNDO.SE*":GOSUB 9100:PRINT
84 S$="CALCULA QUE EL VIAJE DURARA OCHO*":GOSUB 9100:PRINT
85 S$="SEMANAS, PERO PODRIA DURAR MAS. DEBES*":GOSUB
   9100:PRINT
86 S$="CONTRATAR UNA TRIPULACION, PAGARLES,
   COMPRAR*":GOSUB 9100:PRINT
87 S$="PROVISIONES, EQUIPO Y MERCANCIAS*":GOSUB 9100:PRINT
88 S$="PARA COMERCIAR. DISPONES DE 2000 PIEZAS*":GOSUB
   9100:PRINT
89 S$="DE ORO PARA GASTOS.*":GOSUB 9100:PRINT:GOSUB 9200
90 PRINT:S$="      BUENA SUERTE!*":GOSUB 9100:GOSUB 9200:PRINT
91 S$=K$:GOSUB 9100
94 GET P$:IF P$=""THEN 94
95 GOSUB 9200
```

Esta sección llama a las subrutinas que permiten que el jugador contrate una tripulación, provea al barco de provisiones para el viaje y adquiera las mercancías a comerciar en el Nuevo Mundo

```
500 GOSUB 1000
550 GOSUB 2000
600 GOSUB 3000
605 REM *** LISTO PARA EMPEZAR ****
610 PRINT CHR$(147)
615 S$="AHORA ESTAS LISTO PARA EMPEZAR*":GOSUB 9100
625 S$="EL VIAJE.*":GOSUB 9100
630 GOSUB 9200
635 PRINT:S$="CUENTAS CON LA SIGUIENTE TRIPULACION:*":GOSUB
   9100
640 GOSUB 9200
645 FOR T=1 TO 5
650 IF CC(T)=0 THEN 670
655 PRINT CC(T);
```

```
660 PRINT C$(T);
662 IF CC(T)=1 THEN PRINT"    ":GOTO 668
664 PRINT "S"
668 GOSUB 9200
670 NEXT
674 GOSUB 9200
675 PRINT:S$="Y LAS SIGUIENTES PROVISIONES:*":GOSUB 9100
680 GOSUB 9200
685 FOR T=1 TO 4
690 IF PA(T)=0 THEN 710
695 PRINT PA(T);U$(T);"S DE";
700 PRINT P$(T)
708 GOSUB 9200
710 NEXT
715 GOSUB 9200
720 PRINT:S$="  TAMBIEN POSEES:*":GOSUB 9100
725 GOSUB 9200
730 IF OA(1)=0 THEN 740
733 IF OA(1)=1 THEN S$="FRASCO DE MEDICINA"*:GOSUB 735
734 S$="FRASCOS DE MEDICINA*"
735 PRINT OA(1);:GOSUB 9100
736 GOSUB 9200
740 IF OA(2)=0 THEN 750
743 IF OA(2)=1 THEN S$="ARMA*":GOTO 745
744 S$="ARMAS*"
745 PRINT OA(2);:GOSUB 9100
746 GOSUB 9200
750 IF OA(3)=0 THEN 760
753 IF OA(3)=1 THEN S$="BOLSA DE SAL*":GOTO 755
754 S$="BOLSAS DE SAL*"
755 PRINT OA(3);:GOSUB 9100
756 GOSUB 9200
760 IF OA(4)=0 THEN 770
763 IF OA(4)=1 THEN S$="BALA DE TELA*":GOTO 765
764 S$="BALAS DE TELA*"
765 PRINT OA(4);:GOSUB 9100
766 GOSUB 9200
770 IF OA(5)=0 THEN 780
773 IF OA(5)=1 THEN S$="CUCHILLO*":GOTO 775
774 S$="CUCHILLOS*"
775 PRINT OA(5);: GOSUB 9100
776 GOSUB 9200
780 IF OA(6)=0 THEN 790
783 IF OA(6)=1 THEN S$="JOYA*":GOTO 785
784 S$="JOYAS*"
785 PRINT OA(6);:GOSUB 9100
786 GOSUB 9200
790 GOSUB 9200
792 PRINT:PRINT"TE QUEDAN ";MO;" PIEZAS DE ORO"
796 GOSUB 9200
797 S$="PULSA CUALQUIER TECLA PARA COMENZAR EL VIAJE*"
798 GOSUB 9100
799 GET P$:IF P$=""THEN 799
800 WT=0:REM PONER A CERO TOTAL SALARIOS
801 HS="N"  :REM INDICADOR DE MEDIA RACION
802 DIM HR(4):HR(1)=1:HR(2)=1:HR(3)=1:HR(4)=1
```

Aquí comienza el bucle principal, utilizando WK para descontar las semanas transcurridas

```
820 FOR WK=1 TO JL : REM BUCLE PRINCIPAL DEL VIAJE
825 GOSUB 4000:REM INFORME ESTADO TRIPULACION
830 GOSUB 4200:REM INFORME PROVISIONES
835 GOSUB 4300:REM INFORME OTRAS MERCANCIAS
840 GOSUB 9200:PRINT CHR$(147)
842 PRINT:PRINT:PRINT
843 S$="SE CALCULA QUE EL VIAJE*":GOSUB 9100
844 PRINT"DURARA AUN OTRAS";INT(JL-WK+1);SEMANAS    "
845 GOSUB 9200
846 PRINT:S$=K$:GOSUB 9100
847 GET I$:IF I$="" THEN 847
850 GOSUB 5000:REM COMPROBAR FACTURA SALARIAL
855 GOSUB 5100:REM REPARTIR RACIONES
860 GOSUB 5500
861 REM GO TO GENERAR EVENTO AL AZAR
870 GOSUB 6500:REM IR A EVENTO MAYOR
875 IF HR(3)=.5 AND RND(1)<.5 THEN PRINT CHR$(147):GOSUB 6050
878 REM ALBATROS SI ESCASEA LA CARNE
879 GOSUB 7200
880 GOSUB 5300:REM INFORME DE FINAL DE SEMANA
889 NEXT WK
890 REM LLEGADA AL NUEVO MUNDO
891 GOSUB 10000
892 GOSUB 10070
893 GOSUB 10300
894 GOSUB 10500
999 END
```

Aquí termina el programa principal. El resto del programa está escrito en forma de subrutinas a las que se llama desde esta sección principal

```
1000 PRINT CHR$(147):PRINT"ETAPA 1 — CONTRATAR LA TRIPULACION"
1010 PRINT " ---------------------- "
1012 PRINT
1015 GOSUB 9200
1020 PRINT:PRINT"CATEGORIAS DE TRIPULANTES DISPONIBLES:"
1025 GOSUB 9200
1030 PRINT
1040 PRINT  "CATEGORIA  DESCRIPCION  SALARIO SEMANAL"
1050 PRINT  "----------  -----------  ----------------"
1060 PRINT "          1   MARINERO          10 PIEZAS DE ORO"
1070 PRINT "          2   MEDICO            25 PIEZAS DE ORO"
1080 PRINT "          3   MECANICO          15 PIEZAS DE ORO"
1090 PRINT "          4   OFICIAL           20 PIEZAS DE ORO"
1100 PRINT "          5   COCINERO          15 PIEZAS DE ORO"
1105 GOSUB 9200
1110 PRINT:PRINT:PRINT
1120 S$="ENTRE LA CATEGORIA DE TRIPULANTE
     REQUERIDA(1-5)*":GOSUB 9100
1122 S$="0 'F' PARA FINALIZAR LA CONTRATACION*":GOSUB
     9100:PRINT:INPUT P$
1125 CT=VAL(P$)
1128 IF LEFT$(P$,1)="F"THEN PRINT:PRINT"FIN DE LA CONTRATACION DE
     TRIPULACION.":GOSUB 9200:GOTO 1310
1130 IF CT>0 AND CT<6 THEN 1150
1139 PRINT:PRINT
1140 PRINT P$;:S$="  NO ES UNA CATEGORIA DE TRIPULANTE*":GOSUB
     9100
1142 GOSUB 9200
1145 S$="INTENTELO NUEVAMENTE, POR FAVOR"
1146 GOSUB 9100
1147 GOTO 1300
1150 PRINT:PRINT
1155 CN=CN+1: REM TRIPULACION CONTRATADA HASTA AHORA
1156 TS(CN,1)=CT:REM CATEGORIA DE TRIPULANTE
1157 TS(CN,2)=100:REM FORTALEZA INICIAL
1158 WT=WT+WG(T):REM SALARIOS TOTALES
1159 CC(CT)=CC(CT)+1:REM CONTADOR CATEGORIAS DE TRIPULACION
1160 S$="TRIPULACION HASTA EL MOMENTO:"
1170 FOR T=1 TO 5
1180 PRINT S$;CC(T);"  ";C$(T);
1185 IF CC(T)>1 OR CC(T)=0THEN PRINT "S":GOTO 1189
1186 PRINT"    "
1189 S$="                "
1190 NEXT
1195 PRINT:PRINT"FACTURA SALARIAL SEMANAL TOTAL  ";WT
1200 IF CN=CM-1THEN PRINT:S$="SOLO UN TRIPULANTE MAS*":GOSUB
     9100: GOTO 1295
1202 IF CN=CM THEN PRINT:S$="    EL BARCO YA ESTA
     COMPLETO!!":GOSUB 9100:GOTO 1310
1295 REM
1300 GOTO 1015
1310 PRINT:S$=K$:GOSUB 9100:PRINT: GOSUB 9200
1320 GET P$:IF P$="" THEN 1320
1999 RETURN
2005 PRINT CHR$(147): REM LIMPIAR PANTALLA
2010 S$="          ETAPA 2 — APROVISIONAMIENTO*"
2015 GOSUB 9100
2020 S$="          ---------- ----------*"
2025 GOSUB 9100
2030 GOSUB 9200:PRINT
2040 PRINT"HAS CONTRATADO UNA TRIPULACION COMPUESTA
     POR";CN;"MIEMBROS."
2045 GOSUB 9200:GOSUB 9200
2050 FOR T=1 TO 4
2055 PRINT
2060 PRINT"CADA MIEMBRO DE LA TRIPULACION NECESITARA  "
2070 PRINT"  AL MENOS  ";PN(T);"  ";U$(T);
2075 IF PN(T)=1 THEN PRINT"  ";:GOTO 2085
2080 PRINT"S";
2085 PRINT"  DE  ";P$(T)
2086 PRINT"A  ";PC(T);"  PIEZAS DE ORO POR  ";U$(T)
2090 PRINT"POR CADA SEMANA QUE DURE EL VIAJE."
2095 GOSUB 9200:PRINT:GOSUB 9200
2100 PRINT"CUANTOS  ";U$(T);"S DE  ";P$(T)
2110 S$="DESEAS COMPRAR*":GOSUB 9100
2120 PRINT
2130 INPUT P$
2140 PA(T)=VAL(P$):GOSUB 9200
2150 IF PA(T)>((CN*8*PN(T))-1)THEN 2260
2160 IF PA(T)=0 THEN PRINT"SI NO COMPRAS NADA DE":GOTO 2180
2170 PRINT"SI SOLO COMPRAS  ";PA(T);U$(T);
2175 IF PA(T)=1 THEN PRINT"  DE":GOTO 2180
2176 PRINT"S DE"
2180 PRINT P$(T);", ":GOSUB 9200
2190 PRINT"ALGUIEN PODRIA PASAR  ";
2200 S$="HAMBRE"
2210 IF T=4 THEN S$="SED"
2220 PRINT S$;"!":GOSUB 9200
2230 S$="QUIERES VOLVER A PROBAR (S/N)":GOSUB 9100
2240 INPUT P$:P$=LEFT$(P$,1)
2242 IF P$<>"S" AND P$<>"N" THEN 2230
2245 IF P$="N" THEN 2400
2250 PA(T)=0:T=T-1:GOTO 2410
2260 IF PA(T)*PC(T)>MO THEN 2270
2265 GOTO 2400
2270 S$="NO TIENES DINERO SUFICIENTE PARA":GOSUB 9100
2280 PRINT PA(T)
2290 PRINT U$(T);"S DE  ";P$(T):GOSUB 9200
2300 S$="POR FAVOR VUELVE A PROBAR*":GOSUB
     9100:PA(T)=0:T=T-1:GOTO 2410
2400 MO=MO-(PA(T)*PC(T))
2410 PRINT:S$="PROVISIONES HASTA AHORA:*":GOSUB 9100
2412 GOSUB 9200
2415 FOR TT=1 TO 4
2420 PRINT PA(TT);U$(TT)
2430 IF PA(TT)=1 THEN PRINT"  DE  ";:GOTO 2440
2435 PRINT"S DE  ";
2440 PRINT P$(TT)
2450 GOSUB 9200
2460 NEXT
2480 PRINT"DINERO QUE TIENES AUN=  ";MO;"  PIEZAS DE ORO"
```

```
2485 GOSUB 9200:GOSUB 9200
2490 NEXT T
2500 GOSUB 9200:PRINT:S$="FIN D_L APROVISIONAMIENTO*":GOSUB
     9100: GOSUB 9200
2510 PRINT:S$=K$:GOSUB 9100:PRINT : GOSUB 9200
2520 GET P$:IF P$="" THEN 2520
2999 RETURN
3000 REM **** ETAPA 3 — OTRAS MERCANCIAS ****
3001 PRINT CHR$(147):REM ETAPA 3
3002 GOSUB 9200
3005 PRINT"        ETAPA 3 — OTRAS  MERCANCIAS"
3010 PRINT"        -------  -----  ----------"
3020 GOSUB 9200
3025 PRINT
3030 S$="HAY OTRAS COSAS QUE PODRIAN*":GOSUB 9100
3035 S$="SERLE UTILES PARA EL VIAJE, POR *":GOSUB 9100
3040 S$="EJEMPLO, MEDICINAS Y MERCANCIAS*":GOSUB 9100
3045 S$="PARA COMERCIAR.*":GOSUB 9100
3046 GOSUB 9200
3050 S$="PUEDES NECESITAR TAMBIEN ESCOPETAS.*":GOSUB 9100
3055 GOSUB 9200:GOSUB 9200
3060 FOR T=1 TO 6
3065 PRINT
3070 PRINT"UN ";D$(T);
3075 S$=" CUESTA*":GOSUB 9100
3080 PRINT OC(T);
3081 PRINT" PIEZA DE ORO";
3085 IF OC(T)=1 THEN PRINT"PIEZA DE ORO":GOTO 3090
3086 PRINT "PIEZAS DE ORO"
3090 GOSUB 9200
3095 S$="TE GUSTARIA COMPRAR (S/N)*":GOSUB 9100
3110 INPUT PS:P$=LEFT$(PS,1)
3115 IF P$<>"S" AND P$<>"N" THEN 3095
3120 IF P$="N" THEN 3175
3125 GOSUB 9200
3130 S$="CUANTO QUIERES*":GOSUB 9100
3135 INPUT P$
3140 TT=VAL(P$)
3145 IF OC(T)*TT>MO THEN 3150
3147 GOTO 3160
3150 S$="NO TIENES DINERO SUFICIENTE*":GOSUB 9100
3152 GOSUB 9200
3154 S$="POR FAVOR VUELVE A ENTRAR*":GOSUB 9100
3155 GOSUB 9200:GOTO 3130
3160 MO=MO-(OC(T)*TT)
3165 OA(T)=TT
3170 GOSUB 9200
3175 PRINT
3176 PRINT"DINERO SOBRANTE= ";MO
3200 GOSUB 9200:NEXT T
3205 GOSUB 9200:PRINT:PRINT
3210 S$="FIN DE LA ETAPA 3*":GOSUB 9100
3220 GOSUB 9200:PRINT
3230 S$=K$:GOSUB 9100
3240 GET P$:IF P$=""THEN 3240
3999 RETURN
```

Desde el bucle principal del viaje se llama a las siguientes subrutinas para analizar el estado actual del barco y la tripulación y confeccionar un informe semanal para el jugador

```
4000 REM INFORME SOBRE EL ESTADO DE LA TRIPULACION
4010 PRINT CHR$(147)
4020 S$="  DIARIO DE NAVEGACION DEL CAPITAN *":GOSUB 9100
4025 S$="  ------------------------------ *":GOSUB 9100
4030 GOSUB 9200
4035 PRINT"AL EMPEZAR LA SEMANA":WK
4040 S$="EL ESTADO DE LA TRIPULACION ES:*":GOSUB 9100
4045 GOSUB 9200:PRINT
4055 PRINT
4060 FOR T=1 TO 16
4070 IF TS(T,1)=0 THEN 4100
4075 PRINT C$(TS(T,1));"      (";
4078 IF TS(T,2)=-999 THEN S$="MUERTO !!!!!!)*":GOTO 4099
4080 IF TS(T,2)>75 THEN S$="MUY SANO*":GOTO 4099
4085 IF TS(T,2)>50 THEN S$="SANO)*":GOTO 4099
4095 IF TS(T,2)>25 THEN S$="ENFERMO !)*":GOTO 4099
4098 S$="MUY ENFERMO !! )*"
4099 GOSUB 9100:GOSUB 9200
4110 NEXT T
4115 GOSUB 9200:PRINT
4119 WW=0
4120 FOR T=1 TO 5
4130 WW=WW+(CC(T)*WG(T))
4135 NEXT
4140 S$="FACTURA SALARIAL PARA LA SEMANA*":GOSUB 9100
4145 PRINT WW;"PIEZAS DE ORO"
4150 GOSUB 9200
4155 WT=WT+WW
4160 S$="TOTAL SALARIOS DEL VIAJE HASTA AHORA:*":GOSUB 9100
4165 PRINT WT;"PIEZAS DE ORO"
4170 GOSUB 9200
4175 PRINT"DINERO RESTANTE =";MO;"PIEZAS DE ORO"
4180 PRINT:S$=K$:GOSUB 9100
4190 GET I$:IF I$=" "THEN 4190
4199 RETURN
4200 REM INFORME SOBRE PROVISIONES
4205 PRINT CHR$(147)
4206 PRINT"EMPEZAR LA SEMANA";WK:GOSUB 9200
4210 S$="TE QUEDAN LAS SIGUIENTES*":GOSUB 9100
4215 S$="PROVISIONES:*":GOSUB 9100
4220 PRINT:GOSUB 9200
4225 FOR T=1 TO 4
4226 IF PA(T)=0 OR PA(T)=-999 THEN 4240
4230 PRINT PA(T);U$=(T);"S DE ";P$(T)
4232 X=INT(PA(T)/(CN*PN(T)))
4235 PRINT"(SUFICIENTE PARA";INT(X);" SEMANAS)"
4239 GOSUB 9200
4240 NEXT
4290 PRINT:S$=K$:GOSUB 9100
4295 GET I$:IF I$=""THEN 4295
4299 RETURN
4300 REM INFORME OTRAS MERCANCIAS
4305 PRINT CHR$(147)
4306 PRINT"AL EMPEZAR LA SEMANA";WK:GOSUB 9200
4310 S$="TAMBIEN POSEES:*":GOSUB 9100
4320 PRINT:GOSUB 9200
4322 IF OA(1)=0 THEN 4332
4325 PRINT OA(1);:S$="FRASCOS DE MEDICINA*":GOSUB 9100
4330 GOSUB 9200
4322 IF OA(2)=0 THEN 4342
4335 PRINT OA(2);:S$="ARMAS*":GOSUB 9100
4340 GOSUB 9200
4342 IF OA(3)=0 THEN 4352
4345 PRINT OA(3);:S$="SACOS DE SAL*":GOSUB 9100
4350 GOSUB 9200
4352 IF OA(4)=0 THEN 4362
4355 PRINT OA(4);:S$="BALAS DE TELA*":GOSUB 9100
4360 GOSUB 9200
4362 IF OA(5)=0 THEN 4372
4365 PRINT OA(5);:S$="CUCHILLOS*":GOSUB 9100
4370 GOSUB 9200
4372 IF OA(6)=0 THEN 4380
4375 PRINT OA(6);:S$="JOYAS*":GOSUB 9100
4380 GOSUB 9200:PRINT
4382 PRINT"TE QUEDAN";MO;:S$=" PIEZAS DE ORO*":GOSUB 9100
4384 GOSUB 9200
4397 PRINT:S$=K$:GOSUB 9100
4398 GET I$:IF I$="" THEN 4398
4399 RETURN
5000 REM COMPROBAR FACTURA SALARIAL .
5005 IF WT>MO THEN 5010
5008 GOTO 5099
5010 PRINT CHR$(147)
5020 PRINT:PRINT:PRINT
5025 S$="ENTRE LA TRIPULACION CORRE EL RUMOR*":GOSUB 9100
5030 S$="DE QUE NO TIENES SUFICIENTE*":GOSUB 9100
5035 S$="ORO PARA PAGARLES CUANDO TERMINE*":GOSUB 9100
5040 GOSUB 9200:PRINT
5050 S$="LOS ANIMOS SE ESTAN EXALTANDO !!*":GOSUB 9100
5055 GOSUB 9200:PRINT
5060 S$="ESPEREMOS QUE CONSIGAS*":GOSUB 9100
5065 S$="OBTENER ALGUN BENEFICIO!*":GOSUB 9100
5066 GOSUB 9200
5070 PRINT:S$=K$:GOSUB 9100
5080 GET I$:IF I$="" THEN 5080
5099 RETURN
5100 REM REPARTIR RACIONES
5103 PRINT CHR$(147)
5105 S$="      REPARTIENDO LAS RAZONES*"GOSUB 9100
5106 S$="      ----------------------*"GOSUB 9100
5107 GOSUB 9200:PRINT"SEMANA";WK:PRINT
5108 H$="N"
5110 FOR T=1 TO 4
5112 HR(T)=1
5115 IF PA(T)>0 THEN 5180
5120 PRINT"NO QUEDA ";P$(T);" !!!!":GOSUB 9200
5130 S$="LA TRIPULACION SE ESTA DEBILITANDO !!*":GOSUB 9100
5135 WF=10:GOSUB 9300
5139 GOTO 5290
5180 X=(PN(T)*CN)*(JL-WK+1)
5185 IF PA(T)<X THEN 5200
5190 GOTO 5270
5200 PRINT"QUEDA POCA ";P$(T)
5205 GOSUB 9200
5210 S$="QUIERES PONER A LA TRIPULACION A*":GOSUB 9100
5215 PRINT"MEDIA RACION DE ";P$(T)
5220 INPUT I$:I$=LEFT$(I$,1)
5221 IF I$<>"S" AND I$<>"N" THEN 5220:REM ERROR EN ENTRADA
5225 IF I$="N" THEN 5270
5230 HR(T)=.5:H$="S"
5240 WF=5:GOSUB 9300
5250 S$="LA TRIPULACION SE ESTA DEBILITANDO!*":GOSUB 9100
5270 X=PN(T)*HR(T)*CN
5272 IF X>PA(T) THEN X=PA(T)
5275 PA(T)=PA(T)-X
5280 IF PA(T)=0 THEN PA(T)=-999
5285 PRINT X;U$(T);"S DE ";P$(T);" REPARTIDAS"
5290 PRINT:GOSUB 9200:NEXT
5295 PRINT:S$=K$:GOSUB 9100
5298 GET I$:IF I$="" THEN 5298
5299 RETURN
```

Ian Mckinnell

Puertos y canales en el Spectrum

Continuando con nuestro análisis del sistema operativo del Spectrum, vamos a ver cómo el ordenador envía los datos a la pantalla y a la impresora ZX por medio de canales

En el Spectrum de Sinclair el medio habitual de entrada de datos es el teclado, y las salidas habituales son una pantalla de televisor o una impresora ZX. Cada uno de estos instrumentos de hardware se denomina *canal* en el sistema del Spectrum. Así, la pantalla es llamada *canal de salida*. Los datos que fluyen desde o hacia el ordenador en forma de caracteres que van a la pantalla o vienen del teclado, son llamados una *corriente (stream)*. Una corriente o flujo de datos puede ser dirigida a través de los diferentes canales, en el supuesto de que el hardware de canales sea capaz de manejar la corriente de una manera correcta.

En el próximo capítulo hablaremos con más detalle de los canales y las corrientes. Mientras tanto, empezaremos por examinar el sistema de E/S, deteniéndonos en la rutina de "salida de un carácter", que se halla en la dirección &0010. El carácter enviado a esta rutina es lanzado a la pantalla o a la impresora, según el canal previamente seleccionado. El dibujo ilustra los diferentes canales de un Spectrum no ampliado y los números de las corrientes asociadas a ellos. Los canales son conocidos con una letra y las corrientes con un número.

En el Spectrum no ampliado sólo actúan las corrientes 0, 1, 2 y 3. El OS dispone las corrientes y los canales conforme al cuadro que vemos en el margen.

Corriente	Canal
0	K
1	K
2	S
3	P

Así, cuando deseamos dar salida a un carácter por un determinado dispositivo habremos de decirle al Spectrum en qué corriente lo deseamos. Para escribir en pantalla, tomaremos la corriente 2, ya que es la que se asocia con el canal S, para lo cual emplearemos una rutina ROM en la dirección &1601 que dirá al OS la corriente que hemos seleccionado. El número de la corriente se coloca en el registro A, antes de que sea llamada la rutina, y ésta abre posteriormente el canal hardware asociado actualmente con dicho número. Por ejemplo, para abrir el canal S para una salida, haremos

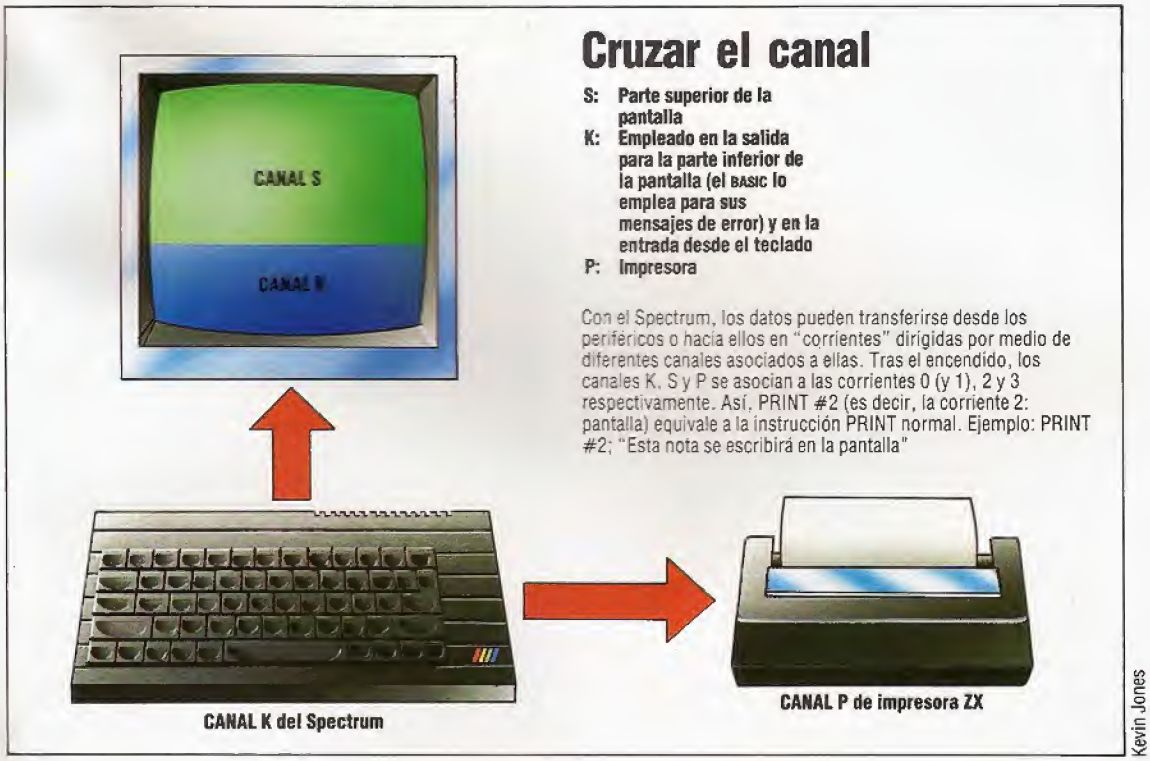
```
LD      A,2
CALL    &$1601
```

Una vez abierto el canal, el envío por él de un carácter se hace simplemente colocando el código del carácter en el registro A y ejecutando después una instrucción RST para llamar a la rutina con dirección &0010.

Cruzar el canal

S: Parte superior de la pantalla
K: Empleado en la salida para la parte inferior de la pantalla (el BASIC lo emplea para sus mensajes de error) y en la entrada desde el teclado
P: Impresora

Con el Spectrum, los datos pueden transferirse desde los periféricos o hacia ellos en "corrientes" dirigidas por medio de diferentes canales asociados a ellas. Tras el encendido, los canales K, S y P se asocian a las corrientes 0 (y 1), 2 y 3 respectivamente. Así, PRINT #2 (es decir, la corriente 2: pantalla) equivale a la instrucción PRINT normal. Ejemplo: PRINT #2; "Esta nota se escribirá en la pantalla"



CANAL K del Spectrum

CANAL P de impresora ZX

Kevin Jones

Esto se parece en algo a la llamada OSWRCH del BBC Micro, salvo que la llamada se hace directamente a la dirección de la ROM y no a través de un vector. El canal S opera sobre el área de la pantalla que es accesible con el PRINT normal del BASIC. Sin embargo también es posible acceder a las dos líneas inferiores que el intérprete del BASIC utiliza habitualmente para dar sus mensajes de error. Como se puede ver en el dibujo, estas líneas forman parte del canal K.

Para enviar un carácter a estas líneas, basta con que abramos el canal K, así:

```
LD      A,0
CALL    &1601
```

Igualmente, el canal de la impresora puede abrirse cargando el registro A con un 3.

No sólo los caracteres habituales pueden enviarse a través de cualquier canal, sino también los caracteres de control. El efecto resultante de estos últimos caracteres dependerá del canal empleado, pero esto significa que podemos obtener las equivalencias en código máquina de PRINT AT, PRINT INK, PRINT PAPER, etc. El cuadro que sigue muestra algunos códigos de control útiles y lo que realizan al ser pasados por el canal S o el K. Es claro que varios de ellos no tendrán efecto alguno si se envían a la impresora ZX (por medio del canal P).

Cod.	Parámetros	Efecto
8	—	Cursor en espacio a la izquierda
9	—	Cursor en espacio a la derecha
10	—	Cursor una línea hacia abajo
11	—	Cursor una línea hacia arriba
12	—	Delete (borrar)
13	—	ENTER
16	n	INK n (necesita un byte más)
17	n	PAPER n (neces. un byte más)
18	n	FLASH n (necesita un byte más)
19	n	BRIGHF n (neces. un byte más)
20	n	INVERSE n (nec. un byte más)
21	n	OVER n (necesita un byte más)
22	nn	AT y,x (necesita dos bytes más) para las coordenadas *x* e *y*)
23	n	TAB n (nec. un parámetro más)

Los bytes requeridos de más por algunos códigos de control son los parámetros que normalmente acompañan a éstos en una instrucción del BASIC. Así, por ejemplo, para ejecutar una orden PAPER 3, enviaríamos simplemente los bytes 17 y 3 al canal S. Este fragmento de programa muestra el equivalente en código máquina de PRINT AT 10,10;"A".

```
3E02    10    ld    a,2          ;abre canal S
CD0116  20    call  #1601        ;seleccionando corriente 2
3E16    30    ld    a,22         ;distintivo de AT
D7      40    rst   #10          ;rutina de sacar un carácter
3E0A    50    ld    a,10         ;ordenada y
D7      60    rst   #10
3E0A    70    ld    a,10         ;abscisa x
D7      80    rst   #10
3E41    90    ld    a,65         código ASCII de "A"
D7      100   rst   #10
C9      110   ret
```

No es difícil, que digamos. Una salvedad notable en esta lista de códigos de control la constituye el código para CLS que no está. Para borrar la pantalla debemos llamar a toda una rutina ROM de dirección &06DB. Es esencial que el canal S esté abierto antes de llamarla, y es también necesario que reabramos el canal S después de haberlo usado si deseamos que imprima cualquier otra cosa en la pantalla. Esta rutina borrará la pantalla:

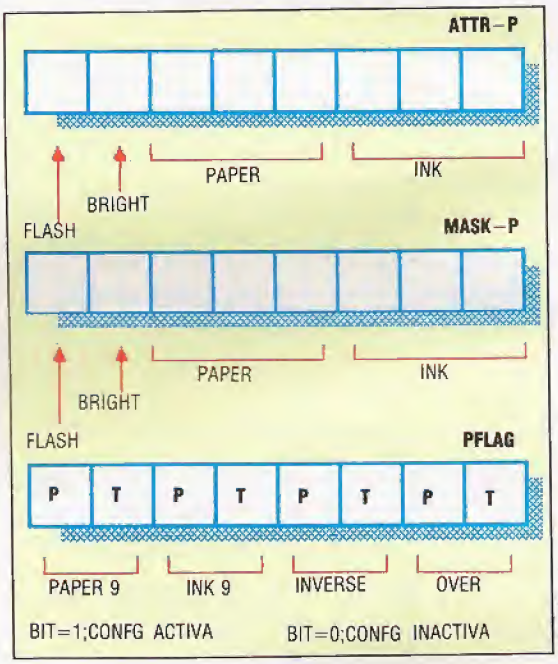
```
3E02    10    ld    a,2
CD0116  20    call  #1601        ;abre canal S
CDDB06  30    call  #06db        ;borra pantalla
3E02    40    ld    a,2
CD0116  50    call  #1601        ;reabre canal S
C9      60    ret
```

Una cualidad útil de la rutina "salida de un carácter" es que los números pasados a ella que representan instrucciones del BASIC son transmitidos por la rutina e impresos completamente. Así:

```
LD      A,249
RST     &10
```

imprimirá, si se seleccionó el canal S, P o K, la instrucción RANDOMISE en la pantalla.

Las instrucciones para gráficos, como PAPER, INK y BRIGHT, realizadas a través de la rutina "salida de un carácter" son sólo operativas para la secuencia de salida de caracteres (se dice que son *ítems temporales del color*). La instrucción PAPER n, fuera de una sentencia PRINT, es *permanente* y opera hasta que se emite otra instrucción PAPER.



Mike Clowes

Las dos variables de sistema que nos interesan son ATTR–P y MASK–P. El esquema anterior muestra cómo estas variables controlan diferentes aspectos de la visualización. En ATTR–P los tres bits que controlan el color de PAPER y de INK de manera permanente se dan, con sus valores respectivos, en el cuadro de la página siguiente, encima de la primera columna de texto. Si el bit de FLASH o BRIGHT es uno, la configuración es operativa. La variable ATTR–P se halla en la dirección 23693.

MASK–P tiene de dirección 23694 y todo bit puesto a uno en este byte hace que los atributos en pantalla situados en la posición relevante de impresión no se alteren con el contenido de ATTR–P. Otra variable importante es la PFLAG que está en la dirección 23697. Damos también un cuadro sobre

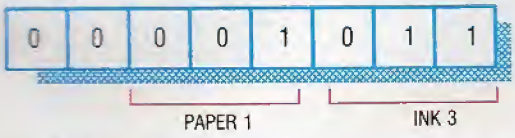
Bits	Color
000	Negro
001	Azul
010	Rojo
011	Magenta
100	Verde
101	Cyan
110	Amarillo
111	Blanco

esta variable. El OS la usa para indicar PAPER 9, INK 9, INVERSE y OVER.

Hay además dos variables en las direcciones 23695 y 23696 y son ATTR–T y MASK–T. Se disponen de modo semejante a las variables ATTR–P y MASK–P, pero controlan los colores temporales (los usados en las sentencias PRINT y en los establecidos enviando códigos de control por medio de la rutina RTS &10).

Para establecer colores mediante ATTR–P y MASK–P basta con manipular el contenido de las variables de sistema, alterando sólo los bits que nos interesen.

Esto se puede realizar fácilmente en el Z80 mediante las operaciones lógicas AND y OR. Así, para ejecutar una instrucción permanente PAPER 1:INK3, estableceremos la variable ATTR–P de la siguiente manera:



mediante este código:

```
LD      A,11
LD      (23693),A
RET
```

Naturalmente, como ocurre con las instrucciones permanentes PAPER e INK del BASIC, los nuevos colores no afectarán a lo que ha sido impreso antes y la pantalla no irá al nuevo color PAPER mientras no se ejecute una instrucción CLS o equivalente.

Las rutinas de gráficos en el Spectrum forman parte del programa intérprete del BASIC. Veamos ahora las operaciones PLOT y DRAW. Al emplear estas rutinas no es difícil realizar cambios de colores alterando las variables ATTR–P y MASK–P, según acabamos de describir.

Las rutinas en sí son fáciles de emplear. La primera, PLOT, se llama en la dirección &22E5. Sus coordenadas le son pasadas mediante el par de registros BC (B para la ordenada *y*, C para la abscisa *x*). Así, para ejecutar PLOT 100,100 desde un programa en código máquina simplemente ejecutaremos lo siguiente:

```
LD      B,100   ;ordenada y
LD      C,100   ;abscisa x
CALL    &22E5   ;lo ejecuta
RET
```

Los cambios de color se realizan con facilidad. La siguiente rutina traza un punto rojo en la pantalla y después restaura ATTR–P a su estado previo antes de volver al BASIC.

```
3A8D5C  10      ld      a,(23693)       ;toma ATTR–P del reg A
F5      20      push    af              ;y lo lleva a la pila
E6F8    30      and     248             ;pone los 3 bits inf a 0
F602    40      or      2               ;pone bit 1 para tinta roja
0664    50      ld      b,100           ;ordenada y
0E64    60      ld      c,100           ;abscisa x
CDE522  70      call    #22e5           ;llama a PLOT
F1      80      pop     af              ;restaura el contenido
328D5C  90      ld      (23693),a       ;original de ATTR–P
C9      100     ret
```

Hasta ahora nos hemos centrado en el canal S. ¿Qué decir del canal K y sus facilidades de salida? Si se quiere, es posible escribir en la parte inferior de la pantalla, pero si el OS o el intérprete genera un mensaje, éste se sobreimprime. Si empleamos este canal para entradas, no se necesita ni siquiera llamar a la rutina de ROM. El teclado es inspeccionado cada 20 microsegundos y se afectan algunas variables de sistema según se haya pulsado o no alguna tecla. Dos ejemplos de empleos son LAST–K (con dirección, 23560), que retiene el código del carácter correspondiente a la tecla últimamente pulsada, y una variable de sistema con dirección 23556, que retiene el valor 255 si en ese momento *no* se ha pulsado tecla alguna. Se puede usar la rutina para que espere hasta que se pulse una tecla y después lleve el código del carácter al registro A. Consiste sencillamente en comprobar el contenido de la dirección 23556 y ver cuándo no es 255. El valor contenido en LAST–K será en ese momento el de la tecla recién pulsada.

```
                10 ;rutina OBTENCION CARAC
3A045C          20 key: ld      a,(23556)       ;comprueba si se pulsó
FEFF            30      cp      255             ;alguna tecla
28F9            40      jr      z,key           ;siguen comprobando
3A085C          50      ld      a,(23560)       ;lleva LAST–K al reg A
C9              60      ret
```

Antes de abandonar el teclado, repasemos un par de variables de sistema de bastante utilidad.

Variable	Dirección	Descripción
REPDEL	23561	Retardo de un 50avo de segundo antes de que el teclado comience repetición. Se puede alterar este valor y el retardo antes de que el teclado inicie la repetición
REPPER	23562	Retardo entre la segunda repetición del teclado y las siguientes. También se puede alterar (POKE)
PIP	23609	Duración del sonido que acompaña la tecla pulsada. Un valor mayor genera un *pitido*

Otro canal del Spectrum no ampliado que nos será útil es el canal P, empleado por la impresora ZX. Tiene habitualmente 3 corrientes asociadas. De este modo:

```
LD    A,3
CALL &1601
```

abrirá el canal P y si se conecta la impresora ZX se imprimirán los caracteres que sigan. Está claro que el BASIC hace un uso muy amplio de los diferentes canales: PRINT y LIST utilizan el canal S, LPRINT y LLIST el canal P e INPUT utiliza el canal K. Podemos añadir nuestros propios canales a estos tres que acabamos de examinar, de manera que podamos acceder con facilidad a algún dispositivo adicional como el microdrive, otras impresoras u otras configuraciones hardware.

En el próximo capítulo examinaremos este punto con mayor atención.

Tacto vital

El control del cursor mediante la selección de iconos le confiere a "Shadowfire" una excitante dimensión

Shadowfire: Para Commodore 64 y Sinclair Spectrum (ambas versiones en la misma cassette)
Editado por: Beyond Software, Competition House, Farndon Road, Market Harborough, LE16 9NR, Gran Bretaña
Autores: S. Cain, D. Colcough, K. Davies, G. Everett, J. Gibson, F. Gray, J. Heap, A. Noble y C. Parrott
Palanca de mando: Opcional
Formato: Cassette

Hasta el observador más indiferente se habrá percatado ya de que los juegos por ordenador se están volviendo cada vez más sofisticados. En lugar de reproducir servilmente juegos de estilo recreativo o de limitarse a juegos de aventuras basados sólo en texto tomados del formato de *Calabozos y dragones*, los diseñadores de juegos para ordenadores personales están desarrollando su propio estilo. Estos juegos combinan muchas de las características de los formatos recreativo y de estrategia para producir una diversión que dura bastante más de los cinco minutos más o menos que dura la acción recreativa, y que es más exigente incluso que los intelectualmente complejos enigmas de los juegos de aventuras.

El argumento de *Shadowfire* básicamente es el siguiente: el general Zoff, un desertor, ha capturado al embajador Kryxix, quien posee los planos para un nuevo tipo de nave espacial denominada *Shadowfire*. El jugador ha de rescatar el embajador antes de que éste se vea forzado a revelar los planes, y dispone de 100 minutos para cumplir su misión. Para ayudarlo a realizar el rescate cuenta con seis personajes, cada uno de los cuales posee diferentes puntos fuertes y débiles.

Para introducirse a bordo de la nave espacial de Zoff, su equipo debe ser "teletransportado". El único miembro capaz de organizar esto es el zángano Manto, de modo que primero debe ser enviado él para que tienda el haz teletransportador que seguirán los demás.

Sin embargo, antes de que usted los despache, es aconsejable proveer a cada personaje de las armas disponibles, en función de sus cualidades y su "talón de Aquiles".

Shadowfire utiliza un sistema exclusivo para desplazar los personajes, permitiéndoles recoger objetos y luchar. A diferencia de otros programas de aventuras, que exigen que el usuario digite instrucciones tales como "avanzar norte" o "recoger láser", este juego permite llevar a cabo todas las acciones mediante iconos y un cursor móvil: algo así como el sistema operativo empleado en el Apple Macintosh. Por ejemplo, para proporcionar granadas de mano al líder del equipo, Zark, usted debe seleccionar el icono de éste. Tras elegirlo, la pantalla pasa a una visualización gráfica de su fuerza, su estamina y otros atributos. En el lado derecho de la pantalla hay tres iconos "monitores" que representan el movimiento, la modalidad de batalla y la pantalla de objetos.

Seleccionando la pantalla de objetos con el cursor (que se puede mover desde el teclado, mediante la palanca de mando o un lápiz óptico), ésta cambia otra vez, pasando a visualizar los objetos que se hallan en las inmediaciones del personaje, así como varios iconos de "actividades". Escogiendo el icono "recoger" y desplazando después el cursor hasta el icono de la granada, se equipará a Zark con granadas.

Una vez armado y "teletransportado" hasta la nave espacial, el equipo comenzará a buscar al general Zoff y al embajador Kryxix. La nave espacial se compone de numerosas habitaciones y pasillos, algunos de los cuales contienen armas o llaves que le permitirán abrir puertas cerradas, mientras que otros ocultan guardias enemigos, a los que habrá de destruir antes de poder seguir adelante. Si usted carece de la llave apropiada, puede requerir los servicios de Sevrina, quien se especializa en hacer saltar cerraduras.

Para emplear de la mejor forma posible los 100 minutos de que dispone, debe trazar una estrategia. Algunos personajes se desenvuelven mejor que otros en ciertas situaciones y, por lo tanto, es recomendable tener a la persona correcta en el lugar adecuado y en el momento oportuno.

Lo que hace de *Shadowfire* un juego tan interesante es el control por cursor. Con el proceso de selección de iconos, el jugador puede reaccionar ante las situaciones con mayor rapidez que si hubiera de digitar cada instrucción por separado.

Los protagonistas de la acción Vemos aquí, en situaciones diferentes, a tres de los personajes que el jugador tiene bajo su control en *Shadowfire*. A la izquierda, Torik se halla en modalidad de movimiento, con el sombreado indicando las posibles direcciones que puede tomar. En el centro, Zark Mondor es atacado y se halla en modalidad de ataque. Nuevamente, se destacan las posibles direcciones que puede tomar y, a la izquierda de la pantalla, aparecen sus adversarios en forma de iconos. Maul, un droide de batalla, aparece con una pantalla de objetos. La sección del centro de la parte inferior de la visualización muestra los objetos que lleva actualmente consigo, y a la izquierda aparecen los objetos que se hallan cerca de él y que puede recoger. Las instrucciones se le envían al ordenador desplazando un cursor (una cruz blanca) hasta el icono seleccionado y pulsando el botón de disparo de la palanca de mando

Aprender a aprender

Para resolver muchos de los problemas que se presentan en el campo de la AI es preciso crear un sistema que aprenda por sí solo

Si, con el correr del tiempo, un sistema de ordenador ha perfeccionado su ejecución de una tarea determinada sin ser preprogramado, cabe afirmar que ha aprendido. Es importante destacar que ello supone un criterio acordado, o normalizado, en relación al cual se pueda medir al sistema. En ausencia de una forma acordada de evaluar el progreso, no tiene ningún sentido hablar de aprendizaje. Por consiguiente, un algoritmo de aprendizaje intenta cumplir uno o más de los siguientes cometidos:

- Cubrir una gama más amplia de problemas.
- Ofrecer soluciones más exactas.
- Obtener respuestas a un costo inferior.
- Simplificar el conocimiento codificado.

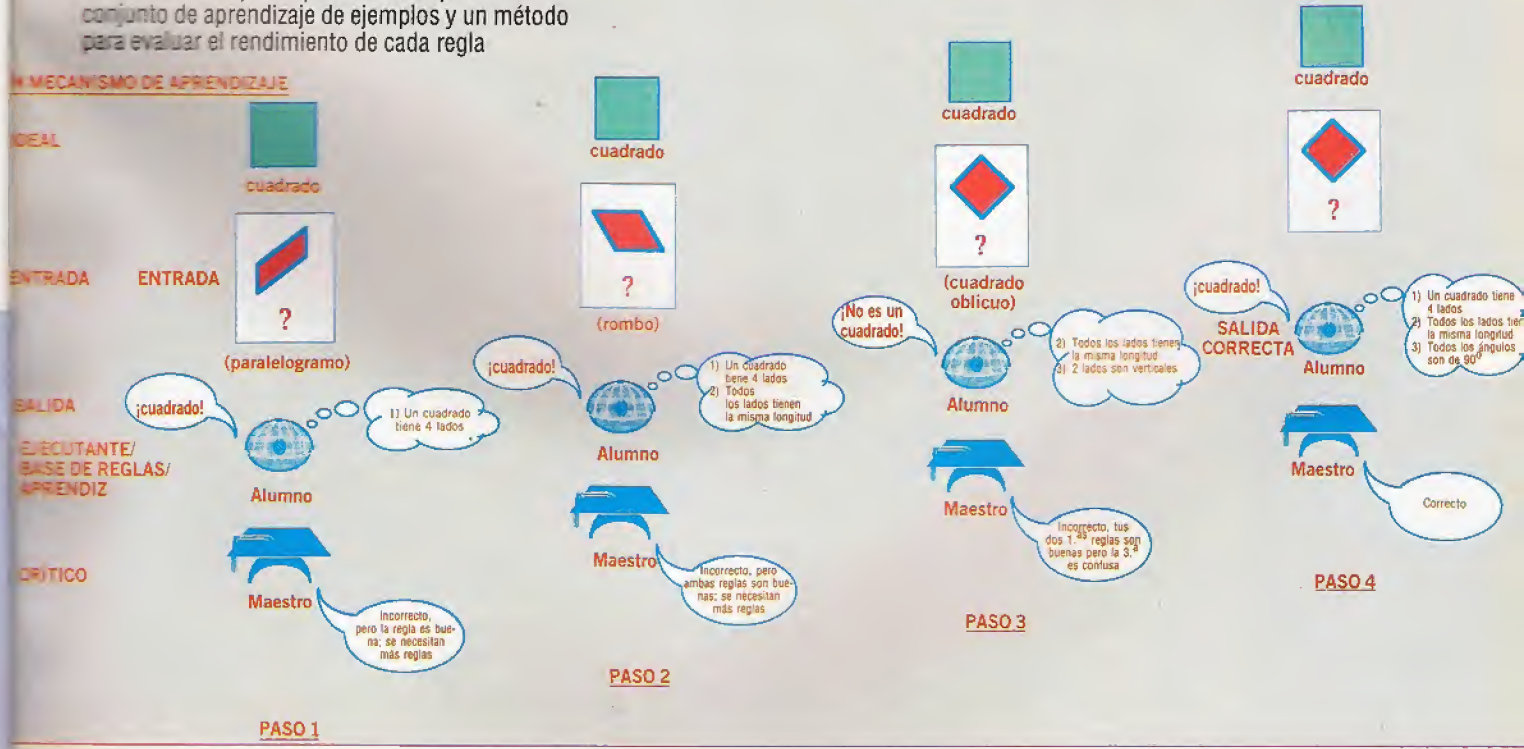
El último punto da por sentado que la simplificación del conocimiento almacenado es valiosa, aun cuando no resulte en perfeccionar el rendimiento de la tarea del ordenador. Ello puede ser así si el sistema comienza con un conjunto de reglas y acaba con otro conjunto, igualmente eficaz, que resulte ser más comprensible para nosotros.

El aprendizaje de la máquina se puede aplicar en muchos campos; pero los sistemas de aprendizaje de mayor éxito se han aplicado a problemas de clasificación.

Ian McKinnell

Inteligencia crítica

Los psicólogos han sugerido que el niño aprende mediante la formación de estructuras de reglas, conocidas como *esquemas*, a menudo mediante un proceso de ensayo y error, comprobando hipótesis (las *reglas*) y conservando aquellas que ofrecen resultados correctos. En muchos casos el proceso de aprendizaje lo dirige alguien que posee un conocimiento sobre los resultados ideales (el *crítico*). El crítico (o maestro) ayuda al niño (el *aprendiz*) a evaluar y refinar su conjunto interno de reglas, que lo ayudará a perfeccionar su rendimiento ante una tarea determinada. Los sistemas de aprendizaje AI intentan reproducir este proceso mediante la creación de una base de conocimientos que se pueda utilizar junto con un conjunto de aprendizaje de ejemplos y un método para evaluar el rendimiento de cada regla



El objetivo de un sistema de este tipo consiste en tomar un dato de entrada y clasificarlo, identificarlo o interpretarlo de alguna manera.

Se han probado diversos procedimientos para obtener mejoras automáticas del rendimiento, siendo Arthur Samuel, con sus clásicos estudios del aprendizaje de la máquina mediante el juego de damas, el pionero de dos de los métodos más simples: el *aprendizaje de rutina* y el *ajuste de parámetros*.

El *ajuste de parámetros*, un método en el cual los coeficientes y los parámetros del programa se ajustan repetidamente al objeto de mejorar el rendimiento, es simplemente una especie de técnica de optimización. Este método se ha estudiado de forma exhaustiva en la literatura sobre matemática aplicada y, por lo tanto, está relativamente bien comprendido. El *aprendizaje de rutina* es completamente "no creativo", y puramente una técnica de compresión del almacenamiento, mientras que los métodos que consideramos aquí son capaces de generalizar y, por tanto, aprender la respuesta apropiada en una situación hasta entonces desconocida.

Todo sistema diseñado para crear nuevos conocimientos y, por tanto, mejorar su rendimiento, debe incluir los siguientes componentes fundamentales:

- Un conjunto de estructuras de datos que represente el actual nivel de pericia del sistema (las *reglas*).
- Un algoritmo de tarea (el *ejecutante*) que utilice las reglas para dirigir la actividad de resolución del problema.
- Un módulo de realimentación (el *crítico*) que compare los resultados actuales con los objetivos deseados.
- El mecanismo de aprendizaje propiamente dicho (el *aprendiz*) que emplee la realimentación del crítico para corregir las reglas.

El método de representación elegido para codificar el conocimiento del sistema es al menos tan importante como los detalles del algoritmo de aprendizaje. En consecuencia, antes de construir un sistema de aprendizaje, es esencial asegurar que el lenguaje de descripción empleado con el sistema sea capaz de expresar las clases de distinciones que serán necesarias, problema nada trivial.

Suponiendo que podamos resolver el problema de idear un lenguaje de descripción adecuado, nos queda el problema de automatizar la generación de descripciones precisas mediante el mismo.

Una forma de enfocar el problema es hacer como si se tratara de una búsqueda a través de todas las descripciones posibles con el fin de hallar aquellas que sean útiles en un contexto dado. Un sistema de AI que aprenda a clasificar puede comenzar con una serie de parámetros que se utilizarán para efectuar una clasificación. En la fase de aprendizaje, el sistema podrá generar y evaluar otras descripciones mediante la combinación, de una u otra manera, de los parámetros originales, conservando las descripciones que sean de ayuda para una clasificación correcta y descartando las que no sirvan. La cantidad de descripciones generadas que sean válidas sintácticamente puede ser astronómica, y, cuanto más expresivo sea el lenguaje de descripción, más explosivo será este problema combinatorio.

Es evidente que se debe hallar alguna manera de dirigir la búsqueda ignorando al mismo tiempo la inmensa mayoría de descripciones potenciales que sean irrelevantes.

Numerosos métodos han funcionado bien como casos de aprendizaje "libres de ruido", en los cuales las clasificaciones están bien definidas, con pocas áreas oscuras entre las clases. Sin embargo, tratar con datos "ruidosos" es un problema que plantea más desafíos. Veamos ahora un procedimiento comparativamente simple que parece funcionar bastante bien en este segundo caso.

BEAGLE (*Biological Evolutionary Algorithm Generating Logical Expressions*: algoritmo de evolución biológica que genera expresiones lógicas) es un sistema de ordenador que produce reglas de decisión mediante la inducción a partir de una base de datos. Como tal, direcciona el problema (con frecuencia eludido) de saber de dónde provienen las reglas de un sistema basado en reglas. BEAGLE trabaja según el principio de la "selección natural", en el que las reglas que no se adaptan a los datos se suprimen y se sustituyen por "mutaciones" de reglas mejores, o por reglas nuevas creadas mediante la unión de dos reglas mejor adaptadas. Las reglas son expresiones booleanas representadas mediante estructuras arborescentes.

El software original estaba compuesto por dos programas en PASCAL: HERB (*Heuristic Evolutionary Rule Breeder*: creador de reglas evolucionistas heurísticas) y LEAF (*Logical Evaluator And Forecaster*: evaluador y pronosticador lógico).

HERB exige que el usuario cree tres archivos de entrada: uno de datos, otro de resultados finales y un archivo de reglas viejo, que puede estar vacío. Produce como salida un archivo de reglas nuevo, que es al menos tan bueno como el viejo. El archivo de datos contiene el conjunto de aprendizaje, para el cual se conoce la pertenencia a la categoría correcta. Asimismo, el usuario ha de llenar una matriz de resultado final, que define el valor o costo de clasificaciones correctas e incorrectas.

LEAF es más simple que HERB. Toma sencillamente un archivo de datos del mismo formato que el conjunto de aprendizaje y ejecuta sobre él un archivo de reglas. Se le puede solicitar que imprima, entre otras cosas, una lista ordenada de elementos del archivo de datos de aquellos que tengan mayores probabilidades de encajar en una clase dada (tal como la defina el archivo de reglas) hasta aquellos que menos probabilidades tengan de encajar en la clase.

El algoritmo BEAGLE consiste en repetir el procedimiento siguiente durante un cierto número de "generaciones" (una generación es una pasada completa a través de los datos de aprendizaje).

1. Evaluar cada regla en cada muestra de acuerdo a la matriz de resultado final, dándole una prima a las reglas más cortas.
2. Colocar las reglas por orden descendente de méritos y suprimir la mitad inferior.
3. Reemplazar las reglas "muertas" aplicándole un procedimiento de emparejamiento a un par de supervivientes elegidas al azar, recombinando, por tanto, porciones de reglas buenas.
4. Transformar unas pocas reglas elegidas al azar (pero no la superior) y aplicarles a todas las reglas nuevas un procedimiento TIDY, preparado ya para la próxima generación.

Charles Darwin (1809-1882)
Las teorías de Darwin sobre la evolución sostienen que las especies se adaptan y se perfeccionan a sí mismas en respuesta a su medio ambiente mediante la selección natural, sobreviviendo sólo los miembros más fuertes y mejor adaptados para criar la siguiente generación y transmitirle sus características. Esta idea se ha empleado con total éxito en el campo del aprendizaje de la máquina, permitiendo que los sistemas de AI mejoren su rendimiento en la clasificación de problemas mediante el desarrollo de conjuntos de reglas de clasificación. No es mera coincidencia el hecho de que uno de tales sistemas se denomine BEAGLE, nombre del barco de Darwin en su famosa expedición a las islas Galápagos

BBC Hulton Picture Library

El procedimiento TIDY reduce ciertas redundancias sintácticas (en inglés) que se puedan haber generado, como dobles negaciones y expresiones constantes, dejando el árbol de reglas "podado" esencialmente con el mismo conjunto de reglas, con la excepción que estará expresado de forma más sucinta. Este método para la generación de reglas nuevas se basa en las ideas darwinianas sobre la evolución: para criar a la nueva generación sólo sobreviven las reglas más aptas.

Informe meteorológico

Un ejemplo de sistema de aprendizaje es el que utiliza el registro del tiempo de un día para pronosticar si lloverá al día siguiente. Para determinar esto la máquina debe "aprender" una regla buena. El sistema puede empezar con una serie de características sobre el tiempo: precipitaciones, intensidad del sol, velocidad máxima del viento y presión del aire (todo ello medido por la tarde, pongamos por caso). Para poder aprender las reglas que permitirán que la máquina formule pronósticos adecuados, se debe reunir un conjunto de registros de datos. Por cada registro se sabrá si lloverá al día siguiente. Un registro típico sería:

Lluvia	0
Sol	7.2
Viento	22
Presión	1017

El sistema debe producir reglas (tal vez, al principio, de forma aleatoria) y utilizar el aprendizaje para comprobar su validez. Las reglas se pueden retener como componentes, tales como:

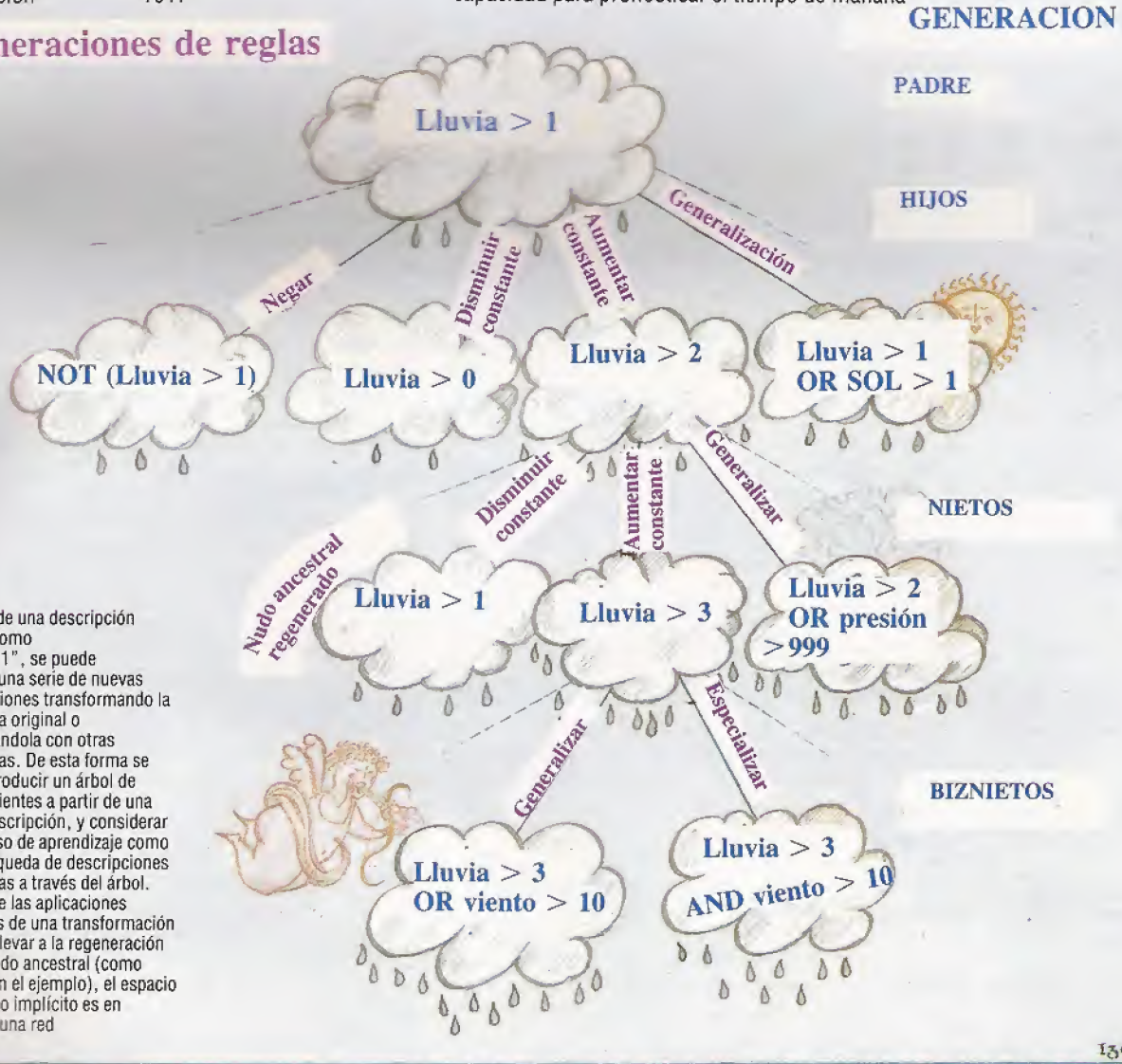
Características	Viento
Constantes	10
Operadores de comparación	> o <
Operadores lógicos	AND, OR y NOT

y éstos combinarse en expresiones como:

sol <4 AND presión <1000

Tras cada pasada de aprendizaje se pueden conservar las reglas buenas y desechar las malas, sustituyendo éstas por nuevas reglas "criadas" con partes de las reglas buenas restantes. Desarrollando reglas que mejoran su rendimiento a cada pasada de entrenamiento, la máquina debe mejorar su capacidad para pronosticar el tiempo de mañana

Generaciones de reglas



A partir de una descripción inicial, como "lluvia>1", se puede generar una serie de nuevas descripciones transformando la sentencia original o combinándola con otras sentencias. De esta forma se puede producir un árbol de descendientes a partir de una única descripción, y considerar el proceso de aprendizaje como una búsqueda de descripciones adecuadas a través del árbol. Dado que las aplicaciones repetidas de una transformación pueden llevar a la regeneración de un nudo ancestral (como vemos en el ejemplo), el espacio de trabajo implícito es en realidad una red

Gabrielle Izen

Réplica perfecta

Al impartir la instrucción SYSGEN, el CP/M se copia a sí mismo de un disco a otro

Hasta ahora en esta serie hemos venido examinando la gama de instrucciones disponibles en CP/M y ya estamos en condiciones de llevar a cabo todas las funciones que se podrían necesitar en casa o en la oficina. Podemos cargar, editar, guardar y transferir los diversos tipos de archivos que están definidos por el CP/M, así como examinar sus propiedades. No obstante, hay una función vital que no hemos realizado hasta el momento: a saber, copiar el propio CP/M.

Como es obvio, es sumamente importante poder transferir el CP/M a otros discos; no sólo para proveernos de una copia de seguridad, sino también porque muchas aplicaciones requieren colocar en el disco una versión del CP/M especialmente instalada. Un ejemplo de ello es el Dr. LOGO en el Amstrad, que incorpora una versión del CP/M hecha a medida en la misma cara que el lenguaje que, a su vez, contiene procedimientos extras para editar y colorear.

Formando pistas

Si bien la mayor parte de un disco de datos CP/M está disponible para archivos del usuario, las pistas más exteriores está reservadas para que las utilice el propio OS. La pista cero retiene el cargador de arranque, que carga automáticamente el CP/M tras el encendido y el reset, mientras que las pistas uno y dos retienen el sistema CP/M propiamente dicho. Tras ellas están las pistas del directorio, que retienen los FCB, que muestran exactamente dónde están situados los registros de que consta cada archivo

Kevin Jones

Copiar el CP/M es bastante simple. En el disco del sistema se proporciona una instrucción transitoria llamada SYSGEN; al ejecutarla, el programa le presentará una serie de preguntas, según las características del sistema, que se pueden utilizar para transferir el CP/M al disco de destino.

Esto suena muy fácil. Sin embargo, la operación plantea algunas cuestiones interesantes sobre la naturaleza del CP/M. ¿Por qué necesitamos una instrucción especial? ¿Por qué en un listado del DIRectorio no aparecen ni el CP/M ni sus instrucciones residentes (incorporadas)? Y, además, ¿cómo lee realmente el CP/M un directorio? Para responder a estas preguntas, centrémonos en la construcción del CP/M.

El OS que se carga en el ordenador se puede dividir en tres partes, cada una de las cuales maneja una parte determinada de la operación de disco. La parte con la que usted estará más familiarizado es el CCP (*Console Command Processor*, procesador de instrucciones de consola), que parece natural puesto que se trata de la "fachada" del sistema, el área que envía información al monitor de video e interpreta las instrucciones que se digitan en el teclado.

Sin embargo, hace muchas más cosas entre bastidores. Cuando se digita información y se entra por el teclado, el CCP da por sentado que la información representa instrucciones y, en consecuencia, éstas se transfieren al área de buffer del CCP a través de otra parte del sistema denominada BIOS (*Basic Input/Output System*: sistema básico de entrada y salida). El CCP realiza, entonces, una comprobación de las instrucciones residentes para ver si su instrucción corresponde a alguna de ellas. De ser así, esa instrucción se ejecutará de inmediato. Si la instrucción no es ninguna de las "incorporadas", el CCP dará por sentado que es transitoria y la buscará en el disco del sistema. Una vez localizada, se cargará en la memoria y se ejecutará. Si el CCP no consigue encontrar la instrucción, generará, por supuesto, un mensaje de error *"file not found"* (archivo no hallado). En el caso del CP/M, el CCP visualizará el nombre de la instrucción en letras mayúsculas, seguidas por un signo de interrogación. La causa de que la instrucción se visualice en mayúsculas es que una de las funciones del CCP es convertir todas las instrucciones en minúsculas a mayúsculas, para adecuarlas al sistema de denominación de archivos del CP/M

Sistema básico de entrada/salida

El BIOS es un conjunto de rutinas (conocidas como activadoras de periféricos) que manipulan todas las funciones de entrada/salida del CP/M. Su finalidad principal consiste en gestionar los periféricos que estén conectados en cada momento al sistema (incluyendo el teclado y la pantalla: el CCP se limita a enviar e interpretar información a través del BIOS). El sistema utiliza parámetros proporcionados por el CCP. Si usted deseara "diseñar a medida" el CP/M para ejecutarlo en otra máquina, es el BIOS lo que tendría que alterar para que el programa se pudiera ejecutar de forma correcta. El motivo de ello (aparte de que el CP/M tenga ciertas exigencias de hardware) es que el método de administración de periféricos puede experimentar grandes variaciones entre máquinas diferentes.

La tercera parte del CP/M es la sección que se ocupa directamente de la gestión del disco. Esta sección se denomina BDOS (*Basic Disk Operating System*: sistema operativo básico de disco). Éste es el componente menos "visible" del CP/M y, sin él, el CP/M sería incapaz de alcanzar su objetivo fundamental. BDOS es el software que administra los archivos en disco, asigna

áreas para almacenamiento y mantiene al día el directorio. A continuación, veamos cómo se distribuye el espacio de almacenamiento en un disco flexible.

Sectores "soft" y "hard"

Desde el punto de vista del hardware, un disco está dividido en 40 pistas. Las versiones modernas del CP/M también tendrán las pistas divididas en *sectores soft* (así llamados porque es el software el que determina los sectores). Los tipos más antiguos de sistemas de disco flexible subdividían las pistas en *sectores hard*, que eran leídos mecánicamente por medio de hardware.

La mayoría de las pistas de un disco flexible se utilizan para almacenar información del usuario; no obstante, hay tres pistas (en un disco de 5 ¼ pulgadas) reservadas para que las utilice el CP/M. Estas pistas exteriores (numeradas del 0 al 2) contienen el breve programa *cargador del arranque* (también conocido como *monitor residente*), que permite que el CP/M se cargue a sí mismo en el ordenador sin que haya ningún sistema operativo de disco presente en memoria. Estas pistas también contienen el sistema CP/M propiamente dicho. Es la información de estas tres pistas la que se instala en el disco mediante la instrucción SYSGEN.

Los sectores (a los que se alude en términos CP/M como *registros*) contienen 128 bytes de información. Se pueden agrupar registros entre sí para formar una *unidad*; una unidad puede contener hasta 128 registros. Un archivo CP/M puede retener hasta 16 de estas unidades. Por lo tanto, un archivo se compone de un máximo de 128 por 128 por 16 bytes (256 Kbytes).

A menos que el archivo sea verdaderamente muy pequeño, no se lo podrá retener en un solo registro. Tampoco será posible, en términos prácticos, acomodar todos los registros de un archivo por orden secuencial en disco. Por consiguiente, un sistema operativo de disco debe poseer algún método para saber qué sectores de un disco se están utilizando para cada uno de los archivos particulares.

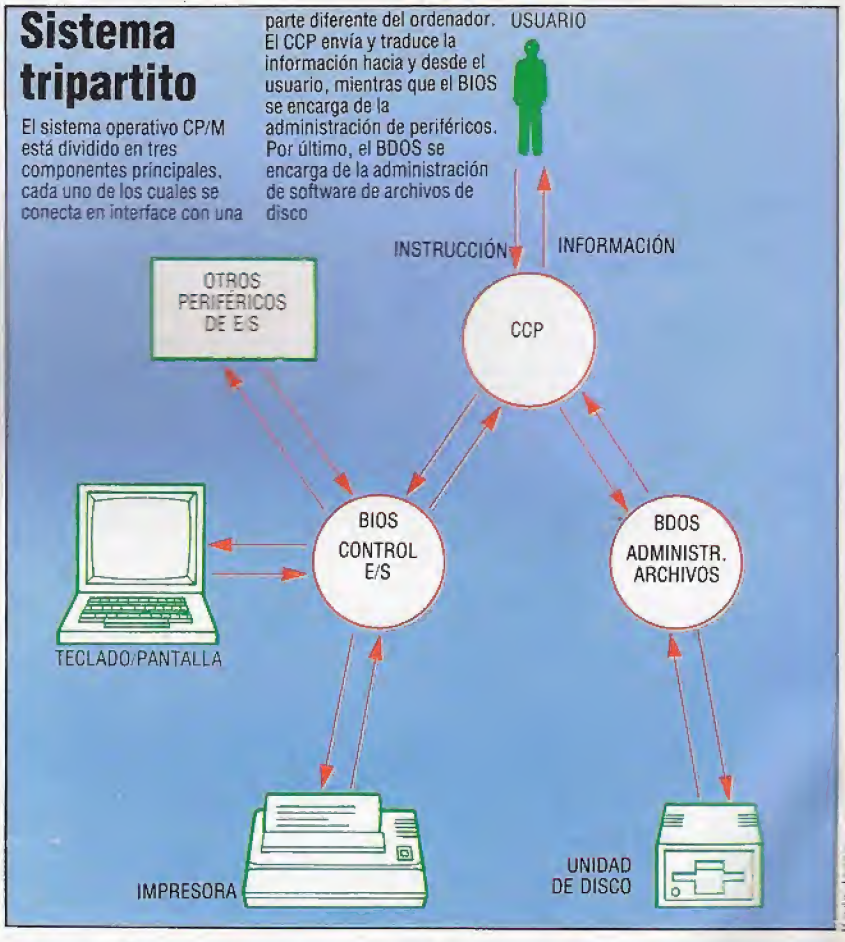
En CP/M, esta información se conoce como FCB (*File Control Block*: bloque de control de archivos). En un disco de 5 ¼ pulgadas, estos FCB están retenidos en la cuarta pista (que se conoce como pista del directorio). Este método de indexación de archivos es leído, escrito, alterado y administrado por el BDOS. Un FCB se compone de hasta 33 bytes de información y retiene todos los datos necesarios para posibilitar que el BDOS identifique y localice cualquier archivo en un disco.

El FCB se compone, en primer lugar, del "tipo de entrada", un único byte que retiene la identidad de la unidad de disco en la cual reside actualmente el disco que contiene el archivo. Los ocho bytes siguientes contienen el nombre del archivo seguido por tres bytes que están reservados para la extensión. Los bytes del 12 al 14 se utilizan para retener la extensión total del archivo, mientras que el byte 15 retiene el contador de registros (que indica cuántos registros tiene el archivo). Los 15 bytes siguientes contienen el mapa de asignación de disco del archivo. Por último, el último byte retendrá el siguiente número de registro a acceder (en caso de que el archivo ocupe más de los 16 K que se pueden describir mediante un solo FCB).

Cuando se ha abierto un archivo mediante la ejecución de una instrucción, el CCP crea una versión del FCB en el área de memoria normalmente reservada para instrucciones transitorias. La información extra acerca del archivo (como su longitud, su cantidad de registros y su asignación de sectores) la proporcionará el BDOS cuando encuentre un archivo cuyo nombre concuerde con el creado por el CCP. La razón de colocar un FCB en la memoria es para que el BDOS pueda actualizar rápidamente los datos retenidos en él mientras las instrucciones del CP/M manipulan el archivo. Finalizadas estas operaciones, el BDOS escribirá en disco la versión final del FCB, teniendo en cuenta la longitud de registros y asignación de sectores alterados.

Ahora podemos retroceder un paso y resumir cómo recupera el CP/M sus archivos. Cuando el CCP recibe una instrucción desde el terminal, interpreta la primera palabra como la instrucción. Si la instrucción no está residente en la memoria, el CCP enviará el nombre del archivo al BDOS a través del sistema de "mensajería" BIOS. El BDOS buscará entonces la pista del directorio del disco hasta hallar una pareja para el nombre de archivo en uno de los FCB. Si el archivo es de instrucciones (COM), el BDOS examinará el mapa de asignación del disco para descubrir qué sectores del disco están ocupados por el archivo; una vez localizados, los copiará en la memoria, en cuyo punto se ejecutará la instrucción.

Esto puede significar que se requiera el BDOS para hallar otro archivo sobre el cual actuar, así como para buscarlo y cargarlo en la memoria. Cuando se carga un archivo en la memoria, el BDOS también proporciona los datos de FCB adecuados a una versión vacía de un FCB que haya reparado el CCP. Este FCB se actualizará a medida que avance la ejecución. Al final del programa de instrucciones, la versión actualizada se volverá a copiar en el disco mediante el BDOS.



Efectos laterales

Veamos cómo el PROLOG incorpora "efectos laterales" y varias características "extralógicas"

El PROLOG, como hemos visto, es un lenguaje esencialmente lógico, pero ciertas funciones que necesita llevar a cabo un lenguaje de programación, como leer y escribir archivos y efectuar cálculos aritméticos, son básicamente procesales y no se ajustan con comodidad a la lógica de predicado. Con el fin de hacer frente a estas funciones, el PROLOG posee una cantidad de predicados incorporados. Algunos de éstos operan en-base a efectos laterales: write(Término), por ejemplo, triunfará siempre, pero, como efecto lateral, llevará a cabo alguna función útil (en este caso, escribir el valor de Término en el canal de salida actual).

En PROLOG, todas las entradas y salidas se manipulan a través de efectos laterales.

Escribir archivos en PROLOG es lo mismo que escribir en la pantalla. Un predicado, tell(Nombrearchivo)(*tell*: decir), le indica al PROLOG que envía toda futura salida al archivo especificado; told (dicho) cierra el archivo. De modo similar, see(Nombrearchivo) (*see*: ver) le dice al PROLOG que lea el archivo mencionado, y seen (visto) restablece el *status quo*. El PROLOG puede enviar códigos ASCII utilizando put(Código) (*put*: poner), mientras que get (obtener) lee valores ASCII.

Para efectuar aritmética, el PROLOG posee varios predicados incorporados que en realidad son funciones que evalúa el intérprete. Uno de tales predicados "evaluables" es is (es). Éste se utiliza de la siguiente manera:

```
C1 is C+1.
```

Que, en BASIC, equivale a:

```
C1=C+1.
```

Observe que C is C+1 siempre fracasará en PROLOG, mientras que el C=C+1 del BASIC funcionará siempre perfectamente. Esto se debe a que lenguajes como el BASIC, FORTRAN, PASCAL, ALGOL y C emplean la asignación destructiva. Es decir, al asignar el resultado de la expresión (a la derecha de una sentencia de asignación) a la variable (a la izquierda), todo valor que esté retenido en esa variable se sobreescribirá.

El PROLOG no utiliza la asignación de ninguna manera. En cambio, emplea la unificación. Esto es, una vez que se ha ejemplificado una variable, toda otra ocurrencia de la misma variable en la totalidad de la cláusula toma el mismo valor. Por tanto, decir que C is C+1 sería como tratar de darle a C dos valores diferentes al mismo tiempo.

El predicado incorporado not (no) se utiliza para introducir información negativa y se define de modo que not(X) será verdadero si X es falso y viceversa, igual que el NOT booleano del BASIC. Para que not(X) sea verdadero, el PROLOG debe primero demostrar que X es falso. Esto lo hace de una forma bastante candorosa: ¡una cosa es falsa si no se puede demostrar que sea verdadera! Imaginemos que tenemos el siguiente programa en PROLOG:

Versión en PROLOG estándar:

```
lenguaje(pascal,difícil)
lenguaje(cobol,difícil)
lenguaje(basic,sencillo)
lenguaje(prolog,sencillo)
```

Versión en MICRO-PROLOG:

```
(lenguaje Pascal difícil)
(lenguaje Cobol difícil)
(lenguaje BASIC sencillo)
(lenguaje Prolog sencillo)
```

que lista unos pocos hechos acerca de la facilidad que entraña la programación en diversos lenguajes. Podríamos formular preguntas tales como:

```
lenguaje(basic,DIFICULTAD).
```

donde DIFICULTAD es una variable, y el PROLOG responderá:

```
DIFICULTAD=sencillo.
```

Pero si preguntamos:

```
lenguaje(basic,fácil).
```

el PROLOG nos responde no. Ello se debe a que el objetivo que hemos planteado no se puede demostrar porque no está ni explícitamente enunciado en la base de datos, ni se puede derivar de otros hechos y reglas. El PROLOG, por supuesto, no comprende el significado de las palabras, de modo que no puede saber que sencillo y fácil significan lo mismo.

Esta idea de la "negación como fallo" es uno de los puntos más débiles del PROLOG y se debe al método de prueba lógica que utiliza el intérprete. Éste es llamado *resolución* y no puede manipular información negativa directamente. Por ejemplo, consideremos esta regla:

```
not(A):-B,C,D
```

(que afirma que X no es el caso si B, C y D son verdaderos). Esto no se puede escribir en PROLOG porque en el encabezamiento de una cláusula no puede haber una negación.

Características extralógicas

El PROLOG no es un lenguaje puramente declarativo. Ya hemos visto cómo el ordenamiento de las cláusulas en el código fuente puede influir en la forma en que funciona el programa, en especial en los procedimientos recursivos. De hecho, la pureza de su lógica se puede diluir aún más mediante el empleo de las llamadas características "extralógicas", fail (fallo) y cut (corte).

La primera de éstas, fail, se puede explicar fácilmente. Se trata de un predicado que fracasará siempre que se lo encuentre. ¿Por qué necesitamos a fail? ¡La razón principal es la de mantener trabajando al PROLOG! Si estamos utilizando nuestro programa de lenguajes y preguntamos:

```
lenguaje(Lang,sencillo).
```

el PROLOG responderá:

```
Lang=basic
```

y se detendrá. Encuentra la primera cláusula de su base de datos que concuerda con el objetivo que hemos planteado y, habiendo demostrado el objetivo, ya no busca más. Pero puede que queramos obtener una lista de todos los lenguajes sencillos. Podemos obligar al PROLOG a producirla si añadimos una regla como:

Versión en PROLOG estándar:

```
listasencillo:-lenguaje(Lang,sencillo),
write(Lang),nl,fail
```

Versión en MICRO-PROLOG:

```
((listasencillo)(lenguaje X sencillo)
(PX)PP FAIL)
```

Ahora, si digitamos listasencillo, el PROLOG replicará:

```
basic
prolog
no
```

Lo que sucedió es que el PROLOG intentó demostrar el objetivo listasencillo demostrando en primer lugar lenguaje(Lang,sencillo). Halló el hecho lenguaje(basic,sencillo) y estableció la variable Lang en el valor basic. Así se demostró el primer subobjetivo. Pasó al siguiente y, con Lang siendo basic, encontró write(basic). Puesto que write triunfa automáticamente (escribiendo basic como un efecto lateral), pasó al subobjetivo siguiente, que es fail. Éste, por supuesto, fracasó, de modo que el PROLOG retrocedió a write. Dado que al retroceder write (y los otros predicados evaluables) tampoco volvieron a triunfar, el PROLOG retrocedió aún más hasta lenguaje(Lang,sencillo) e intentó hallar otra forma de demostrar su veracidad.

La otra característica, cut (¡que se escribe !), también se utiliza para controlar el retroceso. Un cut es un predicado que siempre triunfa. No obstante, como efecto lateral, impide que el intérprete retroceda a través de él. Esto efectivamente congela cualquier opción sobre los valores de variables tomados previamente en un procedimiento. Asimismo, impide que se prueben cláusulas alternativas en el mismo procedimiento. Un uso importante de cut es, en conjunción con fail, asegurar que no se vuelve a probar otra vez una cláusula deliberadamente fallida. Podríamos definir un predicado faltante empleando ! y fail del siguiente modo:

Versión en PROLOG estándar:

```
faltante(Cláusula):-Cláusula,!,fail.
faltante(Cláusula).
```

Versión en MICRO-PROLOG:

```
((faltante X)(?X)/FAIL
(faltante X)
```

que es verdadero si la cláusula dada como su argumento no está incluida en la base de datos, y falso en caso contrario. Así, podríamos preguntar:

```
faltante(lenguaje(basic,sencillo)).
```

Para demostrar faltante(lenguaje(basic, sencillo)), debemos demostrar lenguaje(basic,sencillo). Esto es verdadero porque existe como un hecho en la base de datos. Pasando al cut, éste triunfa automáticamente, pero luego fail hace que fracase toda la cláusula entera.

Sin el cut, el PROLOG habría retrocedido e intentado hallar otra forma de satisfacer Cláusula. No obstante, el cut se levanta como una pared infranqueable en su camino y, dado que no existe ninguna otra forma de satisfacer el objetivo, todo debe fracasar. Asimismo, debido al cut, el PROLOG no buscará cláusulas alternativas, de modo que faltante(Cláusula) acaba siendo falso (porque Cláusula era verdadera) y nosotros no desperdiciamos esfuerzos buscando alternativas.

Este ejemplo en particular ilustra un aspecto muy importante del PROLOG: puede tratar a su propio programa como un dato. Ello hace de este lenguaje una opción ideal para la programación de inteligencia artificial. Éste, junto con otras técnicas de programación avanzada del PROLOG, serán los temas del próximo capítulo.

Kevin Jones

De un lugar a otro
En el último capítulo del PROLOG crearemos un programa para hacer que un robot se desplace por la habitación ilustrada arriba. Sus posiciones se almacenan en el formato lugar(puerta)etc., y el robot se desplaza utilizando ir(Lugar1,Lugar2). Esto supone, en primer lugar, demostrar el subobjetivo lugar(Lugar1), es decir, establecer que el Lugar1 existe y que se puede efectuar un movimiento desde él. ¿Qué otros subobjetivos será necesario demostrar al objeto de planificar un programa de la ruta?

Complementos al PROLOG

El dialecto de PROLOG utilizado a lo largo de esta serie es el denominado DEC-10 PROLOG (en alusión al ordenador de Digital Equipment Corporation en el cual se implementó por primera vez), y está ampliamente reconocido como el estándar. Lamentablemente, la implementación del PROLOG más popular para microordenadores (MICRO-PROLOG) ¡es una de las pocas versiones que difieren de ésta! La otra implementación principal para micros (PROLOG-1) se acerca mucho al estándar DEC-10 (así como su sucesora, PROLOG-2). Los programas en MICRO-PROLOG se escriben enteramente como listas, de forma muy similar a la empleada en LISP. Sin embargo, al programador se le evita el esfuerzo de utilizar esta difícil notación mediante la ejecución de un programa pantalla denominado SIMPLE, que está escrito en MICRO-PROLOG y proporciona una sintaxis alternativa. Existen muchas pequeñas diferencias entre el MICRO-PROLOG y la versión estándar, que hacen que los dos dialectos ofrezcan un aspecto bastante diferente. El MICRO-PROLOG coloca al predicado como el primer elemento de una lista. Por ejemplo:

```
casa_de(marte,marciano).
```

aparecería como:

```
(casa_de Marte Marciano)
```

MICRO-PROLOG también elimina todos los conectores, tales como ':-' y comas, de modo que una cláusula como:

```
casa_de(Planeta,Criatura):-
nacido(Criatura,Ciudad),
en(Ciudad,Planeta).
```

tendría el siguiente aspecto:

```
((casa_de x y)(nacido y z)(en z x))
```

que se puede leer como una lista de tres sublistas, siendo la primera el encabezamiento de la cláusula. El programa *Simple* hace que esta sintaxis sea fácil de comprender y, con su notación de infijos para los predicados, la cláusula podría tener este aspecto:

```
x casa_de y si y nacido z y z en x
```

que sería aún más fácil de comprender si no fuera por el hecho de que el MICRO-PROLOG sólo permite x,y,z,X,Y o Z (o una de éstas seguida por un dígito) como nombres de variables. Entre las otras diferencias se incluye el empleo, por parte del MICRO-PROLOG, de paréntesis en vez de corchetes para las listas; la separación de argumentos y elementos de la lista con espacios y no con comas; la utilización de un '/' en lugar de un '!' para cut; y dar los predicados incorporados en letras mayúsculas. Asimismo, los predicados incorporados poseen nombres no estándares y, en algunos casos, no hacen exactamente las mismas cosas que sus equivalentes estandarizados; pero esto ya es bastante común entre los dialectos de lenguajes de programación. El MICRO-PROLOG sí posee una útil característica de la cual carece el DEC-10 y PROLOG: nos permite crear módulos de programas. Los módulos permiten escribir programas en unidades funcionales y luego combinarlas, sin preocuparse por conflictos entre los nombres de los predicados. Ello le confiere al PROLOG una cierta dosis de estructuración, pero, así y todo, el MICRO-PROLOG aún dista mucho de ser un lenguaje estructurado

Primero de su clase

Con una inmejorable relación calidad/precio, el RM Nimbus irrumpe confiadamente en el mercado de gestión

Research Machines Ltd es un nombre no muy conocido en círculos ajenos al educativo. En los últimos años la empresa se ha establecido como proveedora de microordenadores de ocho bits de gran calidad en escuelas y colegios. El ordenador ampliable 380Z y la máquina en red 480Z causaron un gran impacto desde el momento en que se llevó a cabo en Gran Bretaña el proyecto gubernativo "Los microordenadores en la educación", convirtiéndose en la única competencia seria para el BBC Micro de Acorn.

A pesar del hecho de que ordenadores como el 380Z se consideraron revolucionarios en el momento de su lanzamiento, ahora existe una demanda de mayor potencia de proceso, y esto sólo lo pueden proporcionar los procesadores de 16 bits. En respuesta a esta demanda, Research Machines ha lanzado un nuevo micro de 16 bits bajo el nombre de Nimbus.

A la conquista de un nuevo territorio
El RM Nimbus representa en cierto modo una ruptura para sus fabricantes, Research Machines. Basado en el procesador Intel 80186, capaz de direccionar hasta un megabyte de memoria, la máquina sugiere que la empresa está yendo más allá de su base educativa, establecida ya sólidamente, con las miras puestas en un campo sumamente lucrativo: el mercado de gestión

Chris Stevens

Obviamente, desde que apareciera el 380Z la industria ha experimentado un considerable desarrollo. El extremo superior del mercado en la actualidad está dominado por el IBM PC y toda una multitud de "compatibles", la mayoría de los cuales se basan en el procesador 8088. Aunque el 8088 es más rápido que el Z80, ya está comenzando a parecer algo anticuado. En favor de la empresa se debe reconocer que, al desarrollar el Nimbus, no ha caído en la tentación de producir simplemente otro clono del IBM. En cambio, ha optado por basar la máquina en el chip 80186, mucho más veloz, un auténtico procesador de 16 bits capaz de ejecutar hasta un millón de instrucciones por segundo. El resultado es un ordenador que deja muy rezagada a la mayor parte de la competencia.

Aunque el Nimbus no es compatible con el IBM, el 80186 es un pariente cercano del 8088 y opera bajo el sistema operativo MS-DOS estándar. Asimismo, el teclado también se puede configurar al formato IBM, así como el sistema más familiar de Research Machines utilizado para el 380/480Z. Así, el teclado está equipado con 10 teclas de función programable, un teclado numérico y una letra para ALTernar el juego de caracteres. El tacto de las teclas es excelente en comparación con cualquier otro del mercado.

La versión que presentamos aquí es el Nimbus PC2. Equipado con unidades de disco gemelas de 3 ½ pulgadas formato Sony, existe también una versión con unidad de disco individual, denominada PC1. Cada uno de estos discos puede almacenar hasta 720 K de información y están siendo adoptados por una cantidad creciente de productos de "tecnología punta".

La decisión de incorporar estas unidades, así como otras varias características avanzadas, probablemente haya sido lo que disuadió a RM de producir una máquina compatible con el IBM, en favor de un ordenador más en consonancia con la "tecnología punta". La máquina IBM utiliza el antiguo formato de 5 ¼ pulgadas, que es más delicado que su equivalente Sony y posee una menor capacidad de almacenamiento. Hubiera parecido extraño que RM instalara unidades de disco que están quedando progresivamente anticuadas y que hubieran entorpecido el rendimiento de su máquina, sobre todo si se tienen en cuenta sus otras características avanzadas.

Esto plantea el problema de la compatibilidad de software entre los anteriores ordenadores de Research Machines, que utilizaban discos de 5 ¼ pulgadas, y el Nimbus. La empresa lo ha solventado proporcionando facilidades para permitir la adición de otras unidades de disco, incluyendo, por supuesto, una de 5 ¼ pulgadas. Esto significa que el Nimbus puede leer unidades de 5 ¼ pulgadas en formatos PC-DOS, MS-DOS o CP/M, característica que representa un sólido argumento de venta para todos aquellos usuarios de ordenadores de forma-

tos diferentes que preferirían no renunciar a todo su software actual.

Debajo de las unidades hay dos conectores para cartuchos de ROM. Éstos son compatibles con la puerta para cartuchos instalada en el 480Z, aunque aún es escaso el software disponible basado en ROM. A la izquierda hay un conector que permite la instalación de un *dongle*. Éste es un dispositivo para proteger el software que impide que los usuarios ejecuten un programa sin uno, aun cuando posean una copia del software.

El software que se proporciona con el Nimbus incluye un disco de sistema MS-DOS, el *Wordplan* y *Multiplan* de Microsoft, así como el LOGO y el BASIC propios de RM. Cuando se carga el software, uno comienza a apreciar la excelente máquina que es el Nimbus en realidad. En especial, son sorprendentes las facilidades para gráficos proporcionadas desde BASIC.

El BASIC RM

El BASIC que viene con el Nimbus es un desarrollo de la Version 5 de RM. Teniendo en cuenta que la base fundamental de RM es la del mercado educativo, no es sorprendente que el lenguaje posea varias instrucciones diseñadas para fomentar la programación estructurada, si bien no tantas como la ulterior Version 6 BASIC. El BASIC RM del Nimbus contiene estructuras de instrucción tales como REPEAT..UNTIL, PROC..ENDPROC y GLOBAL, pero carece del posterior DO..WHILE.

Por otra parte, Research Machines ha añadido algunas instrucciones útiles diseñadas para serles de ayuda a los programadores de juegos. Las instrucciones JOYX, JOYY y MOUSE pueden utilizar la palanca de mando y el ratón opcionales; la instrucción BUTTON inicia los efectos producidos por estos dispositivos.

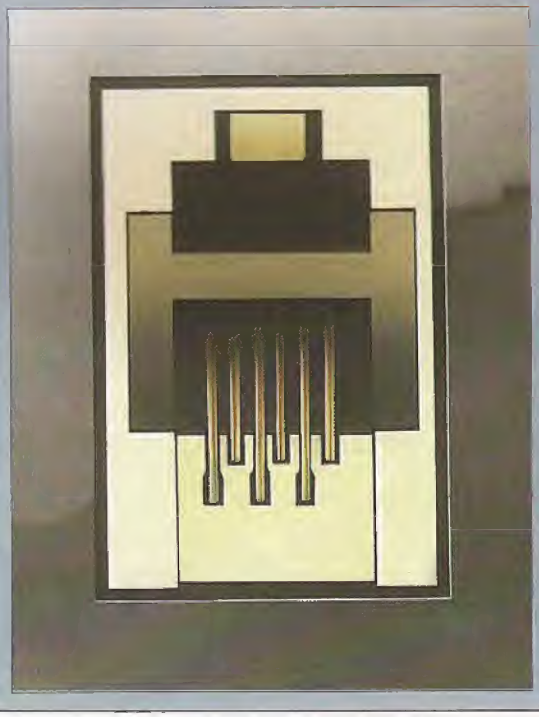
Los gráficos, el sonido y otros atributos se seleccionan desde BASIC utilizando la instrucción SET seguida de los parámetros que se requieran. Mediante el uso de esta instrucción se pueden seleccionar colores de primer plano y fondo (definidos por las instrucciones PAPER, BRUSH y PEN), dibujar líneas y círculos y seleccionar fuentes, direcciones y tamaño de los caracteres. También es posible elegir lo que el manual describe como "estilos" y "patrones vacilantes", produciendo el efecto de un color que es una combinación de los puntos comprendidos en el área.

Aparte de la amplia gama de efectos que se pueden producir en el Nimbus, es notable la velocidad a la cual se dibujan los gráficos en la pantalla desde BASIC, comparable a muchos ordenadores de 16 bits programados en código máquina.

Los discos se organizan en directorios y subdirectorios. Esto significa, por ejemplo, que para poder entrar BASIC, usted debe entrar al directorio de BASIC desde el directorio principal. Esto se realiza desde MS-DOS y el propio BASIC mediante la instrucción CHange DIRectory (cambiar directorio), CD, para más brevedad. Siguiendo esta instrucción, el usuario debe especificar los nombres de "senderos" que se requieren con el fin de obtener el directorio en cuestión. Esto puede sonar un poco complicado, pero permite (y también estimula a ello) organizar los directorios de una forma sistemática. Desde el sistema operativo es posible la creación de nuevos senderos y directorios, además del establecimiento de nuevos senderos entre directorios existentes.

Siendo fiel a su filosofía de diseño para el Nimbus, Research Machines ha optado por la tecnología del "estado del arte" en vez de las puertas para periféricos estándares. El Nimbus posee tres líneas de salida de potencia separadas desde la unidad principal del ordenador. En la esquina superior izquierda de la máquina se halla una salida de 2A desde el transformador principal de potencia, y a lo largo del panel posterior hay dos salidas más pequeñas, de 12 V y 5 V.

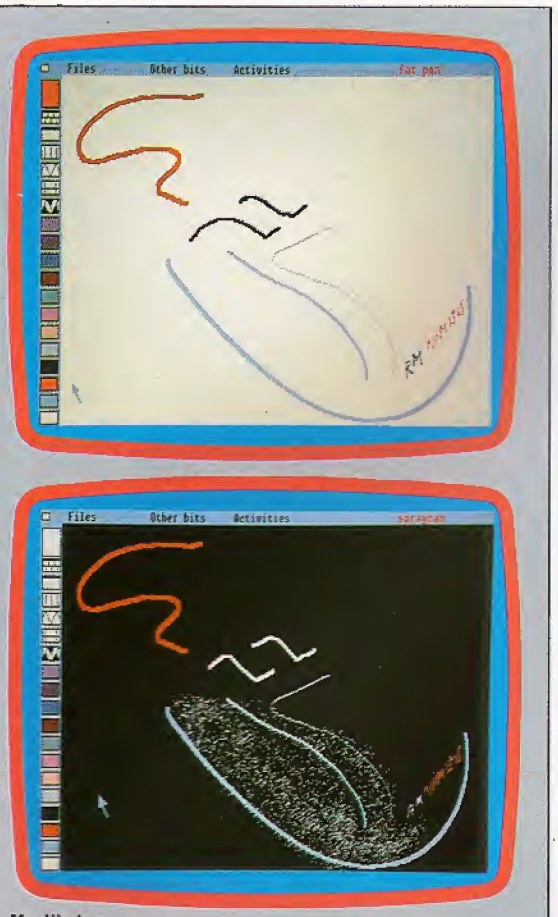
Entre las otras interfaces que se proporcionan hay una puerta Piconet, que se instala como un enchufe telefónico BT estándar. El Piconet permite



Kevin Jones

Ampliación potencial

El RM Nimbus está bien provisto de interfaces. El ordenador posee dos conectores para pantalla, una puerta para ratón/palanca de mando, y una interface Piconet, puerta para impresora y no menos que tres líneas de potencia auxiliares. Las importantes puertas de E/S Piconet y para impresora utilizan el estándar de enchufe BT introducido recientemente, que muchos usuarios con equipos más antiguos no podrán conectar de forma directa. El enchufe de carrillón BT (que vemos a la izquierda) es una interface en serie capaz de manipular señales tanto de entrada como de salida. Hay seis líneas disponibles, que se pueden disponer en numerosas configuraciones para transmisión, interconexión y otras aplicaciones



MacNimbus
Entre los programas de demostración que se proporcionan con el Nimbus hay un paquete de gráficos activado por ratón. Al igual que muchos programas de dibujo por ratón, la pantalla guarda un notable parecido con la del Apple Macintosh. Sin embargo, a diferencia de éste, en la pantalla se pueden visualizar hasta 16 colores distintos al mismo tiempo

Ian McKinnell

gobernar desde una única puerta hasta 30 periféricos separados, cada uno de ellos conectado a través de un módulo externo. La conexión en interface se realiza en serie y está construida de acuerdo al estándar RS422. Es esta nueva conexión de alta velocidad la que permite al Nimbus controlar los 30 dispositivos.

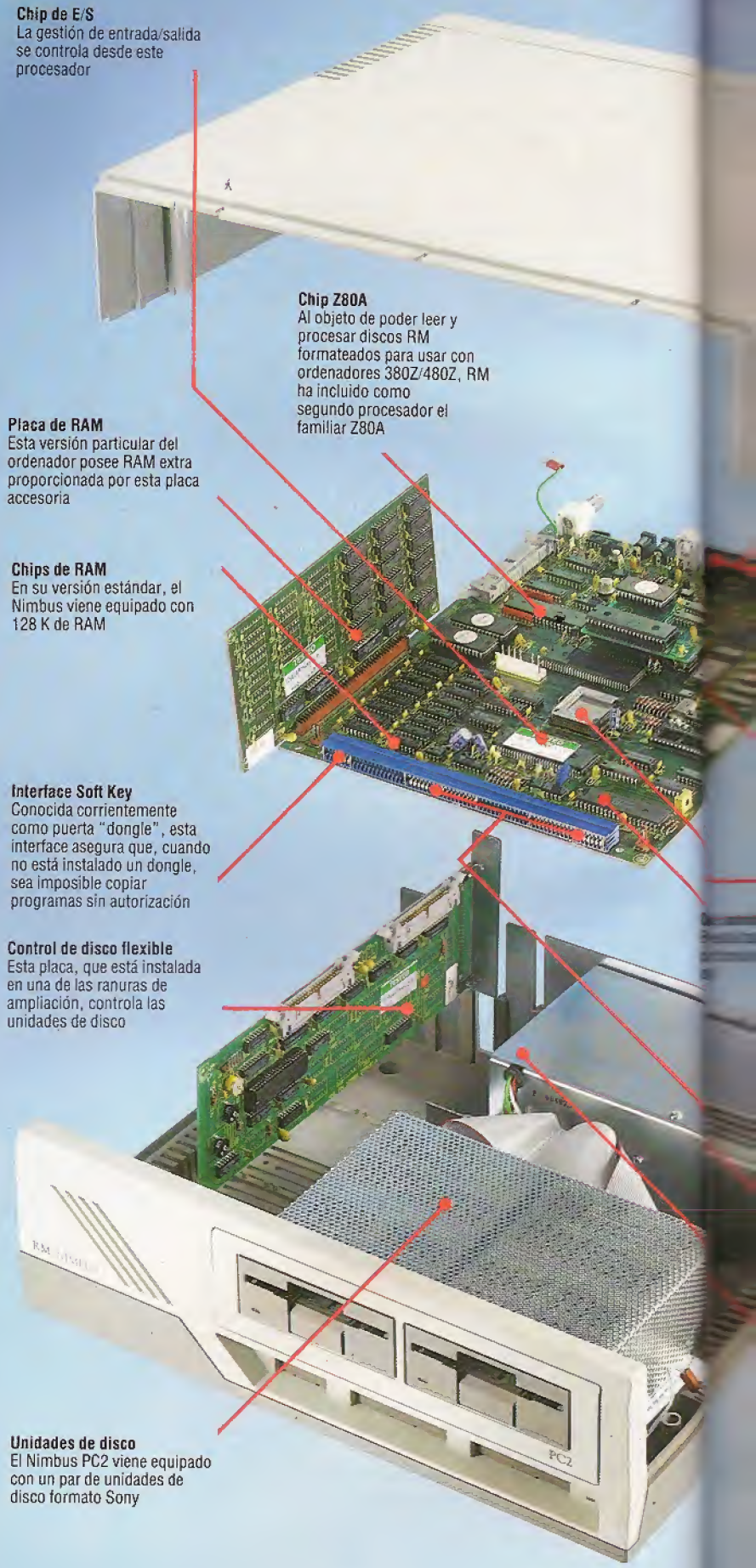
Lamentablemente, en muchos casos el RS422 no es directamente compatible con el anterior estándar RS232, de modo que si usted desea utilizar estos dispositivos habrá de obtener interfaces adecuadas.

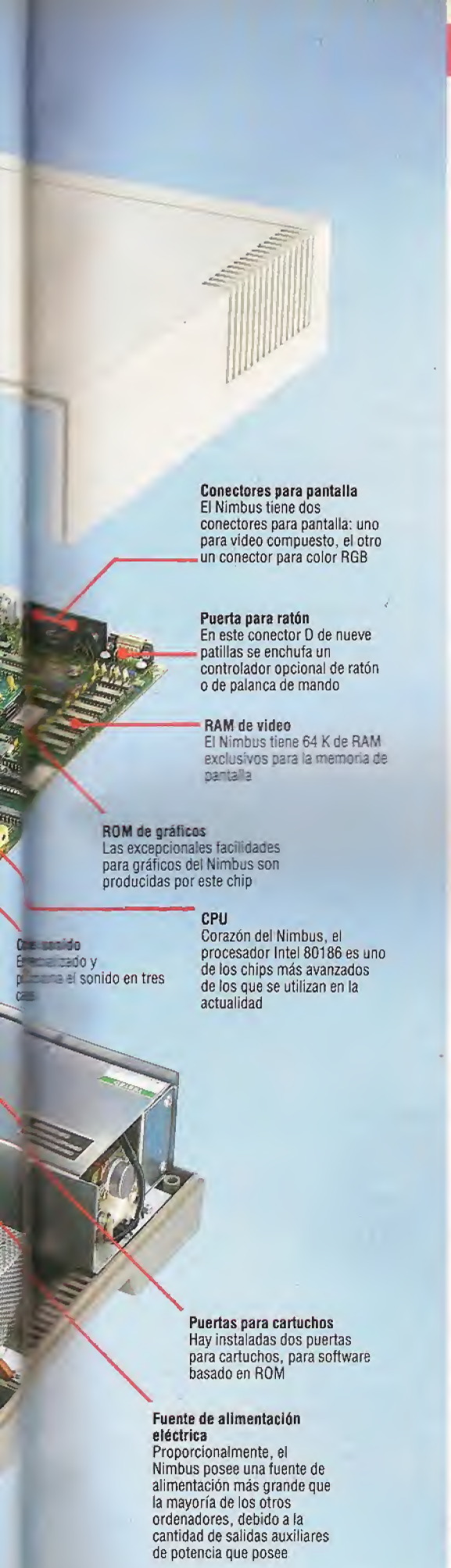
También se proporciona una facilidad de red que permite conectar el Nimbus a una red compuesta por otros ordenadores de Research Machines a través del conector coaxial.

Interface para impresora

La interface para impresora es una inclusión curiosa. Se ha instalado como un enchufe telefónico BT idéntico a la puerta Piconet, y no responde exactamente al estándar. Además, la puerta únicamente es en serie y está diseñada de acuerdo al estándar RS422.

El resto de las interfaces que se proporcionan ya son más familiares. Hay un conector D de nueve patillas que facilita la conexión al ordenador de una





palanca de mando o, más probablemente, un ratón, al objeto de ejecutar aplicaciones tipo Macintosh (los programas de demostración contienen un programa de dibujo que, aparte de los gráficos en color, guarda un notable parecido con la pantalla Macintosh, si bien carece de muchas de las facilidades de *MacPaint*).

Se proporcionan dos conectores para pantalla, uno de los cuales, para monitores RGB, ejecuta la pantalla en color Cub que RM ha diseñado especialmente. El otro es un conector de video compuesto destinado a pantallas monocromáticas. Por último, hay un enchufe DIN para el teclado.

En el extremo izquierdo (mirando hacia la parte posterior de la máquina) hay cuatro puertas de ampliación en las que el usuario puede agregar placas extras. Éstas incluirían bancos adicionales de RAM, interfaces digital/analógico, etc. A diferencia de otros muchos fabricantes que insisten en que esta clase de trabajo sólo lo han de realizar los representantes autorizados, en su guía para el usuario Research Machines ha ofrecido instrucciones completas para que usted instale sus propias placas.

Documentación completa

El manual que se entrega con el Nimbus responde muy bien al elevado estándar que la gente ha llegado a esperar de RM. La información que contiene incluye una guía para principiantes sobre el MS-DOS y los discos, junto con un estudio detallado de las puertas.

Lo único que se echa en falta es un análisis del chip procesador 80816 que, aunque probablemente no fuera de gran utilidad para la inmensa mayoría de usuarios, sí serviría a los programadores que pretendieran desarrollar su propio software en la máquina.

Aún es demasiado pronto para decir a quiénes les resultará más atrayente el Nimbus. Decididamente, es una máquina de una gran calidad que, de entrada, vale casi la mitad que su único rival serio, el IBM PC/AT. No cabe duda de que el ordenador tendrá una buena acogida, pero el volumen de ventas dependerá de la estrategia de comercialización de RM.

Las universidades y otros grandes establecimientos educativos que ya tienen establecidas buenas relaciones con RM son potenciales adquirentes de la máquina, y no se preocuparán demasiado por el apoyo de software: con toda probabilidad, desearán escribir el suyo propio y podrán transferir el software RM que ya posean. Las velocidades y las facilidades para una programación sencilla representarán, obviamente, convincentes argumentos de venta, así como la cantidad de periféricos que se pueden instalar.

Sin embargo, para que el Nimbus satisfaga su potencial en el mercado de gestión (potencial que, evidentemente, está al alcance del ordenador), Research Machines habrá de convencer a los usuarios de gestión de que pondrá a su disposición el software adecuado. RM habrá, asimismo, de adaptar las interfaces al estándar Centronics y RS232 que se utiliza en la mayoría de las oficinas. Si Research Machines consigue introducirse en el sector de gestión, el Nimbus podría convertirse en un importante desafío para los actuales líderes del mercado, ACT/Apricot e IBM.

Chris Stevens

RM NIMBUS

DIMENSIONES
400×350×110 mm

CPU
Intel 80186 operando a 8 MHz

PANTALLA
Máximo de 80×25 caracteres de texto, 640×250 pixels en modalidad a cuatro colores, 320×125 en modalidad a ocho colores

INTERFACES
Enchufe para red, puerta Piconet, puerta para impresora, salidas de potencia de 12 V, 5 V y 2A, conectores para monitor compuesto y RGB

LENGUAJES DISPONIBLES
BASIC RM, LOGO RM y PASCAL RM

TECLADO
83 teclas incluyendo teclado numérico y 10 teclas de función programable

DOCUMENTACIÓN
Los manuales son exhaustivos y contienen la mayor parte de la información que puede necesitar el usuario. Asimismo, parecen ser menos densos y más fáciles de leer que los de la documentación anterior

VENTAJAS
El Nimbus tiene muy pocos competidores a su mismo precio. La tecnología avanzada, junto con su sencillez de uso, excelentes gráficos y buenos manuales, harán que la máquina se convierta en un producto de éxito

DESVENTAJAS
Al menos a corto plazo, quizá la tecnología sea demasiado avanzada; el resto de la industria no está aún a la altura del Nimbus y, por tanto, tal vez haya escasez de ciertas interfaces

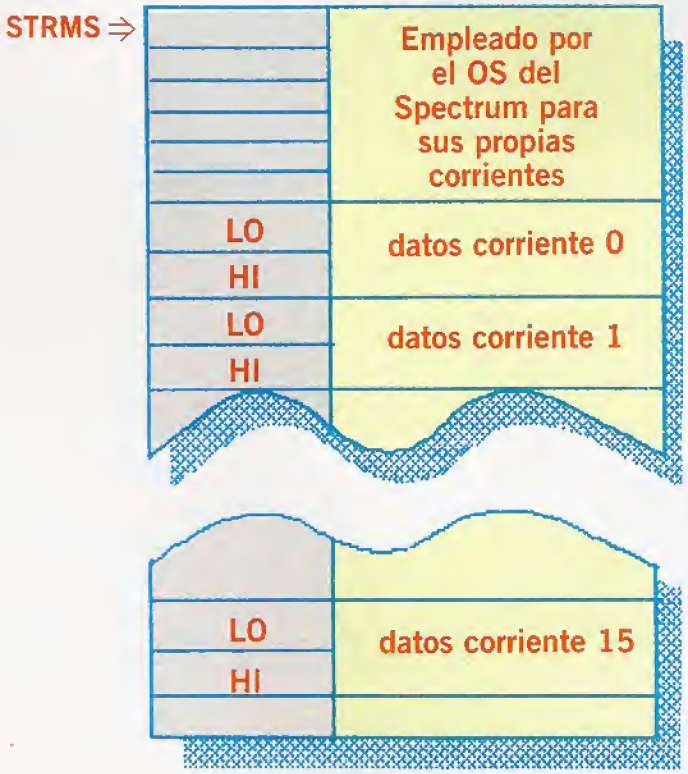
Muy corriente

Veamos ahora cómo fluyen los datos a través de los canales hasta la impresora o la pantalla en un Spectrum

Decíamos que siempre que se enciende o restaura el Spectrum, unas determinadas corrientes se asocian con determinados canales en el proceso de inicialización. Estas asociaciones son las que resumimos en el siguiente cuadro:

Canal	Corriente	Descripción del canal
K	0	Teclado y parte inferior de la pantalla
K	1	
S	2	Parte principal de la pantalla
P	3	Impresora ZX

Sin embargo existen otras corrientes a nuestra disposición no empleadas por el Spectrum normal. Se observará que la siguiente descripción corresponde a un Spectrum sin Interface 1; todo se altera si empleamos redes de área local, microdrive o interface serial.



Estos dispositivos serán analizados más adelante. Las otras corrientes están numeradas del 4 al 15, y pueden también asociarse con cualquier canal de E/S que se necesite intercomunicar.

El OS del Spectrum registra cuál de las corrientes se asocia a cada canal en uso. Esta información se guarda en una tabla de 38 bytes llamadas STRMS (de *streams*: corrientes), que puede encontrarse en el área reservada para las variables de sistema con dirección 23568. Hay seis bytes al comienzo de la tabla empleados por el OS del Spectrum para sus propios fines. Según éstos, cada una de las 16 corrientes está representada por una entrada de dos bytes. La disposición de la tabla es la que muestra el dibujo.

El siguiente fragmento en BASIC imprimirá el contenido de este área de la memoria para que usted pueda conocerlo:

```
10 FOR I=23568 TO 23604 STEP 2
20 PRINT PEEK I+256*PEEK(I+1)
30 NEXT I
```

Nos dará todas las entradas de dos bytes. Las tres primeras son los detalles de las corrientes internas del Spectrum, siguiendo después con las entradas de las corrientes de la 0 a la 15. Los valores en esta última parte de la tabla actúan como punteros de otras entradas en una tabla distinta, situada en cualquier parte de la memoria y a la cual apunta una variable de sistema llamada CHANS. Esta variable se encuentra en la dirección 23631, y la dirección de la tabla de CHANS puede hallarse así:

```
PRINT PEEK 23631+256*PEEK 23632
```

Esta variable de sistema *nunca* será tratada con POKE. La expresión que acabamos de darle proporciona el inicio de la *tabla de registro de canales*, que contiene varios detalles informativos sobre cada canal.

El valor leído de la tabla STRMS sirve de índice de esta tabla de registro de canales. Una entrada de la tabla STRMS da el (desplazamiento + 1) desde el comienzo de la tabla de registro de canales de la información sobre el canal que se está usando con una determinada corriente. Así la entrada de 1 tabla STRMS para corriente 3 (a la impresora) es 16. Si se resta uno de éste obtenemos un desplazamiento de 15. Por lo tanto los datos correspondientes al canal P se encontrarán en la dirección:

```
(PEEK 23631+256*PEEK 23632)+15
```

(siempre suponiendo que se acaba de encender el ordenador y no se han alterado las asociaciones corrientes-canales después de la inicialización).

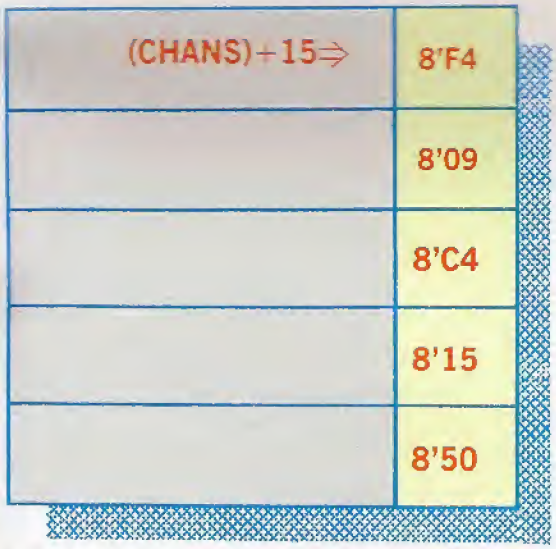
Si los datos que están en una entrada de la tabla STRMS son cero, esto indica que la corriente todavía no ha sido asociada con ningún canal. Éste es el caso, en un Spectrum no ampliado, de las corrientes que van de la 4 a la 15 después de la inicialización.

Cada entrada en la CRT es de la forma:

byte *lo*	Dirección de la rutina de salida para el canal
byte *hi*	
byte *lo*	Dirección de la rutina de entrada para el canal
byte *hi*	
n	Byte único que representa la letra código del canal en ASCII (así, K, S y P)

Una entrada de la tabla STRMS apunta al segundo byte de cada entrada de la tabla de registro de canales (CRT), por eso hay que restar uno antes de su empleo. Esta tabla está en el corazón mismo del sistema de E/S del Spectrum. Si un canal no puede llevar a cabo una operación de E/S, la entrada de la dirección adecuada en la tabla CRT apuntará a una rutina en la ROM que generará el mensaje de error Invalid I/O Device (dispositivo de E/S ilegal). Así, para un canal de sólo salida (como es el canal S) la dirección de la rutina de entrada apuntará a esta rutina de error. En caso contrario, la rutina adecuada de la ROM para salidas o entradas serían las apuntadas por las entradas de la tabla.

Examinemos la entrada del canal P en la tabla. Recuérdese que en circunstancias normales es un canal de sólo salidas, que envía datos a la impresora ZX. Los datos para este canal en la entrada de la CRT son:



La entrada para el canal S se halla en (CHANS)+5, y los datos del canal K están en (CHANS)+0. (CHANS) es la dirección retenida en la variable de sistema CHANS.

Con las instrucciones OPEN# y CLOSE# podemos asociar corrientes no empleadas a uno de los canales que hasta ahora hemos examinado. Pero estas dos instrucciones realmente se utilizan para asociar las corrientes a otros canales en el sistema ampliado, tales como los que ofrece la Interface 1. Incluso de esta manera examinaremos brevemente el empleo de estas instrucciones sin Interface 1. Por ejemplo,

```
OPEN #4,"S"
```

asociará la corriente cuatro al canal S. Si se realiza esta instrucción y después se examina la tabla STRMS, se verá que se ha hecho una entrada en la posición correspondiente a la corriente 4. En la sentencia podemos hacer uso de variables.

```
10 LET n=4
20 LET c$="S"
30 OPEN #n,c$
```

hará exactamente lo mismo que la sentencia anterior. Cuando se abre un canal de esta forma, las instrucciones PRINT# y LIST# le permitirán enviar datos al canal a través de la nueva corriente. Así podemos añadir:

```
40 PRINT #4; "Hola"
```

que se imprimirá en la pantalla. LIST#4 dará un listado del programa que está en pantalla según sabemos. Si una corriente se ha asociado con un canal tanto de entrada como de salida, se puede usar también la instrucción INPUT#. Estas sentencias PRINT e INPUT ampliadas sólo tienen utilidad cuando se puede disponer de los canales que proporciona la Interface 1, como veremos.

Cuando se finaliza con una determinada combinación de corriente y canal, y se desea disolver el emparejamiento, usaremos CLOSE#. Así, en nuestro ejemplo, CLOSE#4 independizará la corriente 4 del canal S. Llegados aquí, todo intento de enviar datos a esta corriente "cerrada" generará un mensaje de error.

Las instrucciones PRINT#,LIST#,INPUT# e INKEY# pueden usarse también con las corrientes de la 0 a la 3.

Estas corrientes están siempre abiertas, como puede comprobar ejecutando lo siguiente:

```
10 CLOSE#2: REM debería cerrar el canal S
20 PRINT#2; "Hola"
```

No aparecerá mensaje alguno de error (el OS se encarga de abrir el canal de nuevo antes de pasarle información). Consideremos ahora la instrucción LIST#.

```
LIST#3
```

equivale, en circunstancias normales, a LLIST: el programa se lista en la corriente 3, la cual, como acabamos de ver, se asocia con el canal P. Las instrucciones LIST#0 y PRINT#0 son bastante interesantes ya que éstas llegan a imprimir en la parte inferior de la pantalla (que normalmente cae fuera del dominio de las sentencias PRINT). Llegados a este punto, adviértase que cualquier texto que se halle impreso en esta parte de la pantalla será sobreescrito por medio de una indicación o mensaje de error del sistema operativo. En circunstancias normales la instrucción PRINT#3 equivale a LPRINT.

Por último, se puede modificar el comportamiento de un canal en el Spectrum para que realice algo diferente de aquello para lo cual fue inicializado. Por ejemplo, el canal P es modificado a menudo para permitir que una impresora Centronics sea controlada por el Spectrum y una interface apropiada. Si alteramos la entrada de la rutina de salida en la entrada adecuada de la CRT, las instrucciones LPRINT y LLIST enviarán datos a una rutina que maneja la impresora Centronics. Naturalmente, no resulta preciso que se trate de una impresora; puede ser una puerta de salida o cualquier dispositivo de salida.

Programa "Imprimir en alt-res"

Este programa intercepta los datos que se envían por el canal P y los desvía a una rutina que permite al usuario la facilidad de imprimir en alta resolución. El listado puede ser entrado mediante un ensamblador corriente (el CHAMP, DevPac o Picturesque, p. ej.). Si usted carece de un ensamblador tendrá que entrarlo en codificación hexadecimal, empleando un monitor o bien mediante el Cargador en Hexa que aquí ofrecemos. Una vez digitado este cargador y ejecutado, se le pedirá que digite la dirección de inicio (que es 65068). Se espera ahora que digite el código hexa, que puede ser entrado sólo mediante dos dígitos a la vez. Por ejemplo, la línea del código:

2130FE

habrá de ser digitada de esta manera:

21 (ENTER)
30 (ENTER)
FE (ENTER)

Entrado el código, falta tan sólo pulsar S para dar por finalizado el programa. Es el momento de la suma de control (que es 29155), que le indicará si cometió algún error. La nueva instrucción que proporciona el programa tiene este formato:

LPRINT AT X,Y;"MENSAJE"

donde X debe valer entre 0 y 168, e Y entre 0 y 248. En vez de "MENSAJE" puede estar una variable en serie o un número. Cuando la serie que se imprime con la nueva instrucción sobrepasa un borde de la pantalla, vuelve a empezar por el comienzo de la misma línea, pero no pasa a una línea más abajo. Las coordenadas X e Y funcionan de idéntico modo que con la instrucción PLOT del Spectrum

Cargador hexa

```
10 CLEAR 63999
15 POKE 23658,8
17 DEF FN H(H$)=16*(CODE
   (H$)-48-7*(H$(1)>"9"))+(CODE
   (H$(2))-48-7*(H$(2)>"9"))
20 INPUT "Dirección inicio"; a
25 LET Z=A
30 INPUT (a), LINE a$
35 IF A$="S" THEN GO TO 90
40 PRINT a,a$: POKE 23692,255
50 FOR Z=1 TO LEN A$/2
60 POKE a,FN H(a$(((z*2)-1) TO)): LET a=a+1
70 NEXT z
80 GO TO 30
90 INPUT "CHECKSUM:";D
100 LET C=0: FOR B=Z TO A-1: LET C=C+PEEK B:
    NEXT B
110 IF C=D THEN PRINT "Codificación perfecta"
120 PRINT "Vaya, la codificación no ha sido correcta"
```

Listado assembly

```
                ORG   65068
        PIXAD   EQU   22AAH
        UDG     EQU   23675
        CHARS   EQU   23606
                ;     "Hace que los datos en CHAN apunten a nuestra rutina"
2A4F5C  ENABL   LD    HL,(23631)
010F00          LD    BC,15
09              ADD   HL,BC
013AFE          LD    BC,DO-IT
71              LD    (HL),C
23              INC   HL
70              LD    (HL),B
C9              RET
                ;     "Guarda regs. y llama a nueva rutina impresora"
E5      DO-IT   PUSH  HL
C5              PUSH  BC
D5              PUSH  DE
F5              PUSH  AF
CD46FE          CALL  DOIT1
F1              POP   AF
D1              POP   DE
C1              POP   BC
E1              POP   HL
C9              RET
                ;     "Comprueba si a contiene el cód. de control de AT"
F5      DOIT1   PUSH  AF
3A1DFF          LD    A,(ATFLG)
FE00            CP    0
200B            JR    NZ,GETXP
F1              POP   AF
FE16    ATCHQ   CP    22
201D            JR    NZ,CRCHQ
3EFF            LD    A,255
321DFF          LD    (ATFLG),A
C9              RET
FEFE    GETXP   CP    254
2809            JR    Z,GETYP
F1              POP   AF
3216FF          LD    (XPOSI),A
211DFF          LD    HL,ATFLG
35              DEC   (HL)
C9              RET
F1      GETYP   POP   AF
3217FF          LD    (YPOSI),A
3E00            LD    A,0
321DFF          LD    (ATFLG),A
C9              RET
                ;     "Comprueba el dato de fin de impresión Z(que aquí es el 13)"
FE0D    CRCHQ   CP    13
2001            JR    NZ,VCHRQ
C9      SKIPC   RET
                ;     "Si el código en a está entre 32 y 128 o si"
                ;     "es un cód. UDG entonces halla su dato en la mem."
FE20    VCHRQ   CP    32
380C            JR    C,PRNT?
FE80            CP    128
381C            JR    C,FCHR
FE90    UDGCQ   CP    144
3804            JR    C,PRNT?
FEA5            CP    165
3804            JR    C,FUDGC
                ;     "Para todos los demás códigos imprimirá un   ?"
3E3F    PRNT?   LD    A,63
1810            JR    FCHR
                ;     "Halla el dato de UDG o de car. en la mem."
D690    FUDGC   SUB   144
210000          LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A7B5C          LD    HL,(UDG)
19              ADD   HL,DE
180C            JR    PRNIT
210000  FCHR    LD    HL,0
6F              LD    L,A
29              ADD   HL,HL
29              ADD   HL,HL
29              ADD   HL,HL
EB              EX    DE,HL
2A365C          LD    HL,(CHARS)
19              ADD   HL,DE
010700  PRNIT   LD    BC,7
09              ADD   HL,BC
221BFF          LD    (CHRAD),HL
                ;     "Comprueba si las posiciones X e Y son correctas"
3A16FF          LD    A,(XPOSI)
FEF9            CP    249
D231FF          JP    NC,ERRB
3A17FF          LD    A,(YPOSI)
FEA9            CP    169
3078            JR    NC,ERR5
ED4B16FF        LD    BC,(XPOSI)
CDAA22          CALL  PIXAD
321AFF          LD    (PIXPO),A
2218FF          LD    (DFADD),HL
                ;     "Toma el primer conjunto de pixels o car. impreso"
0608            LD    B,8
C5      PRNLP   PUSH  BC
2A1BFF          LD    HL,(CHRAD)
7E              LD    A,(HL)
2B              DEC   HL
221BFF          LD    (CHRAD),HL
6F              LD    L,A
                ;     "Si el conj. de pixels primero no ha de moverse"
                ;     "dentro del byte de visual., éste salta adelante"
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAE6FE          JP    Z,PUTIT
                ;     "Mueve el conj. de pixels en los bytes de visual."
                ;     "hacia la posición correcta del pixel"
47              LD    B,A
2600            LD    H,0
CB3D    ROTLP   SRL   L
CB1C            RR    H
A7              AND   A
10F9            DJNZ  ROTLP
                ;     "Lo pone en la posición derecha de la pantalla"
ED5B18FF PUTIT  LD    DE,(DFADD)
1A              LD    A,(DE)
AD              XOR   L
12              LD    (DE),A
3A1AFF          LD    A,(PIXPO)
FE00            CP    0
CAFAFE          JP    Z,PST
13              INC   DE
1A              LD    A,(DE)
AC              XOR   H
12              LD    (DE),A
1B              DEC   DE
2A18FF  PST     LD    HL,(DFADD)
CD1EFF          CALL  ULINE
2218FF          LD    (DFADD),HL
C1              POP   BC
10C4            DJNZ  PRNLP
                ;     "Añade 8 a la posic. X para que el siguiente car."
                ;     "se imprima en espacio borrado"
3A16FF          LD    A,(XPOSI)
C608            ADD   8
                ;     "Si se llega al fin de línea vuelve al"
                ;     "inicio PERO NO baja una línea"
FEF9            CP    249
DA12FF          JP    C,STAIT
3E00            LD    A,0
3216FF  STAIT   LD    (XPOSI),A
C9              RET
00      XPOSI   DEFB  0
00      YPOSI   DEFB  0
0000    DFADD   DEFW  0
00      PIXPO   DEFB  0
0000    CHRAD   DEFW  0
00      ATFLG   DEFB  0
                ;     "Mueve la direc. en HL un pixel arriba de la pant."
F5      ULINE   PUSH  AF
7C              LD    A,H
25              DEC   H
E607            AND   7
200A            JR    NZ,END
7D              LD    A,L
D620            SUB   32
6F              LD    L,A
3804            JR    C,END
7C              LD    A,H
C608            ADD   8
67              LD    H,A
F1      END     POP   AF
C9              RET
                ;     "Crea varios errores"
CF      ERRB    RST   8
0A              DEFB  10
CF      ERR5    RST   8
04              DEFB  4
        FINIS   END
```

Sistema sonoro

Iniciamos un nuevo proyecto: el diseño y la construcción de una interface MIDI para el BBC Micro y el Commodore 64

Las especificaciones de la MIDI (*Musical Instrument Digital Interface*: interface digital para instrumentos musicales) esbozan el hardware y el software necesarios para un sistema que permita que los instrumentos equipados con la interface se comuniquen entre sí. Puesto que la comunicación es de una naturaleza digital, en el sistema MIDI podemos también introducir un micro personal para controlar otros instrumentos o almacenar datos en su memoria, ya sea en cinta o en disco. Las especificaciones de hardware de la interface nos permiten construir una placa de circuitos conectable, fácil de construir y económica.

Una vez construida la interface, se puede programar al ordenador para que interactúe con otros dispositivos del sistema MIDI, escribiendo el software adecuado. Más adelante incluiremos estos listados.

Desde la introducción de las primeras especificaciones MIDI, en agosto de 1983, el mundo de la música electrónica se ha visto inmerso en un ambiente de confusión y malentendidos sobre lo que la interface puede y no puede hacer. Esto se debe a que a los músicos no les preocupa la naturaleza de la comunicación entre los elementos electrónicos, sino que la mayoría de ellos sólo se interesan (y con razón) en lo que un sistema determinado les puede ofrecer en términos de actuación musical.

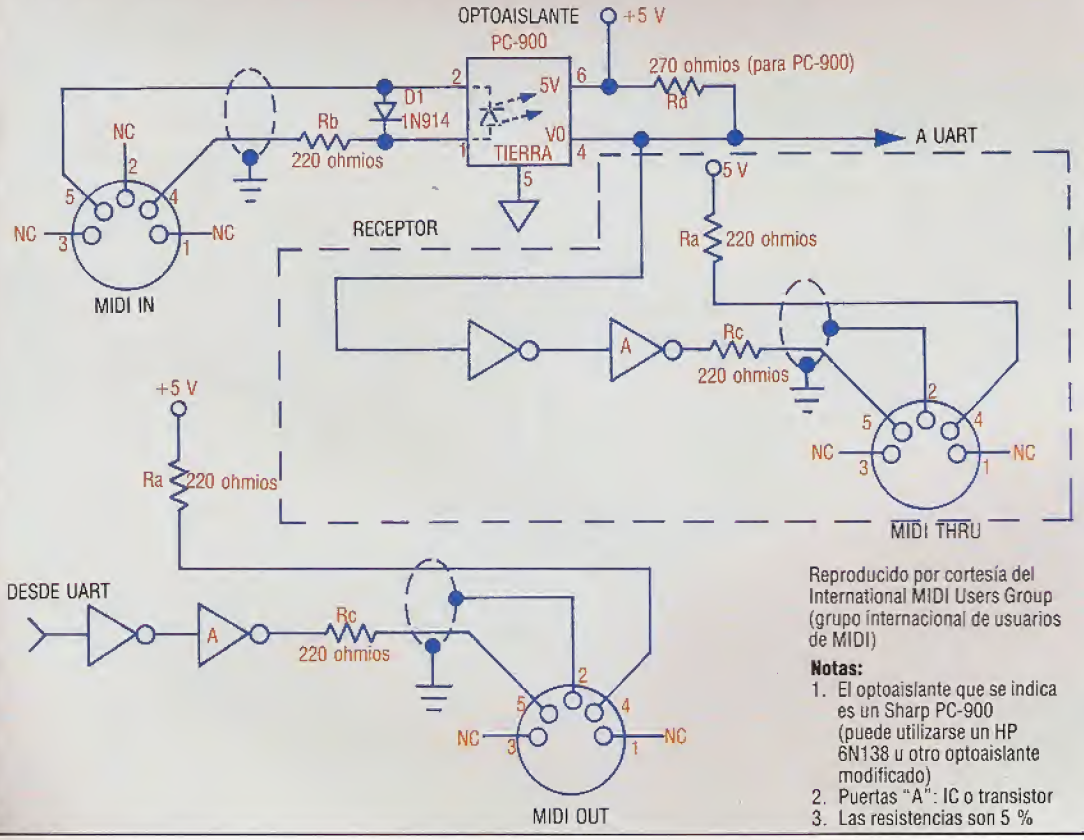
A lo largo del curso explicaremos las especificaciones MIDI con una terminología que les resultará más familiar a los programadores que a los músicos, y les proporcionaremos a usted, de este modo, la misma información básica que utilizan los diseñadores de equipos que pretenden que sus productos sean compatibles con el estándar.

En primer lugar hemos de hacernos alguna idea sobre la naturaleza de los datos comunicados por la MIDI. La mejor manera de entenderlo es considerando las diversas partes de un sistema de música electrónica como los periféricos de un ordenador. Puede ser que la distinción entre estos componentes no se haga evidente de inmediato: por ejemplo, un típico sintetizador polifónico moderno en reali-

Hardware MIDI estándar

Por lo general la interface consta de dos secciones: una para recibir datos y otra para transmitirlos. Cada una posee sus propios conectores, etiquetados MIDI IN y MIDI OUT, respectivamente.

Puede haber un tercer conector, MIDI THRU, que envía una copia de los datos en MIDI IN. Esto es para facilitar la conexión de receptores múltiples en cadena sin la necesidad de múltiples MIDI OUT en el transmisor. Abajo vemos el diagrama de circuitos del hardware de la interface



Reproducido por cortesía del International MIDI Users Group (grupo internacional de usuarios de MIDI)

Notas:

1. El optoaislante que se indica es un Sharp PC-900 (puede utilizarse un HP 6N138 u otro optoaislante modificado)
2. Puertas "A": IC o transistor
3. Las resistencias son 5 %

Kevin Jones

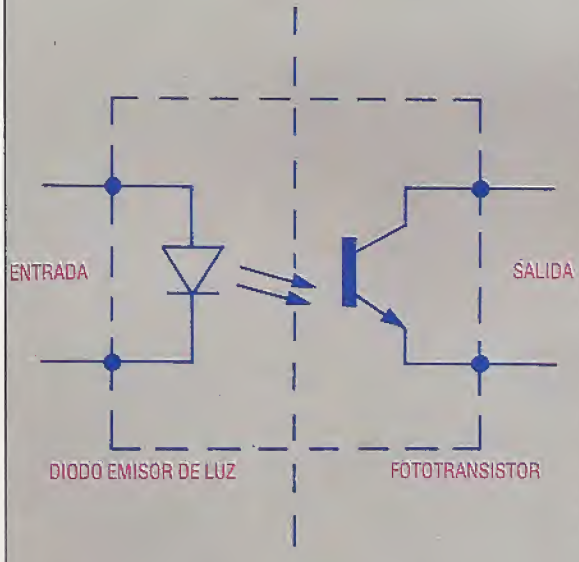
dad está constituido por dos componentes separados, si bien por lo general se hallan contenidos en la misma caja. Los dos componentes son el teclado (el dispositivo de entrada) y el sistema de circuitos para la generación de sonido (el dispositivo de salida). En la modalidad de operación normal, el dispositivo de entrada está conectado directamente con el dispositivo de salida: la pulsación de una tecla produce de inmediato una salida de sonido.

Esta situación es similar a la de una sofisticada máquina de escribir electrónica, en la cual los dos componentes (el teclado y el mecanismo de impresión) se pueden conectar entre sí de modo que los datos de entrada se reproduzcan de forma inmediata. Sin embargo, es muy fácil reconocer el teclado y la impresora como los periféricos potenciales de un ordenador central, que leerá los datos de entrada y los *procesará* antes de enviarlos al dispositivo de salida. Esto, por supuesto, ya nos resulta familiar: se trata de un procesador de textos.

En el caso del sintetizador es importante destacar que la interface (interna) entre el teclado y los generadores de sonido es puramente digital. El teclado actúa simplemente como un controlador del sintetizador, de la misma forma en que el teclado de la máquina de escribir actúa como un controlador para el mecanismo de impresión. Del mismo modo, el enlace se puede romper insertando un ordenador entre los dos dispositivos, formando un sistema *procesador de música*. En este caso la música no se representa como un patrón de variaciones de presión de aire (como el producido por un disco fonográfico o un disco compacto, en el que la música se codifica digitalmente), sino como una secuencia de *eventos* entrados en el teclado musical y transmitidos subsiguientemente al dispositivo de salida. La función de los circuitos para generación de sonido es la de convertir estos eventos en señales electrónicas que se reproducirán como sonido mediante un sistema de altavoz/amplificador. En consecuencia, la MIDI introduce por primera vez la seria perspectiva de controladores que no sean teclados, tales como instrumentos de cuerda o viento, siempre que sus respectivas salidas respondan a las especificaciones MIDI.

Hay un buen ejemplo de sistema para procesamiento de música que, aunque primario, es muy conocido, y se remonta a mucho antes de la edad del ordenador moderno. La pianola era un tipo especial de piano que podía tocar una música grabada en forma de patrón de agujeros en un rollo de papel. Mientras el papel rodaba a través del piano a una velocidad constante, un mecanismo detector captaba la secuencia y combinaciones de agujeros, cada uno de los cuales representaba una nota dada. Los martillos del piano golpeaban entonces las

Circuito del optoaislante



El segundo requisito del hardware MIDI especifica que la línea de entrada debe estar aislada del equipo receptor para evitar bucles a tierra en el sistema. Éstos se deben evitar porque tienden a manifestarse en forma de un zumbido muy audible a la frecuencia de la red eléctrica (50 Hz). Este problema se puede superar mediante un dispositivo denominado *optoacoplador* u *optoaislante*. Se trata de un circuito integrado con una conexión óptica (y no eléctrica) entre su entrada y su salida (ver diagrama). Las dos patillas de entrada del chip optoaislante están conectadas a un LED interno, similar a las luces indicadoras rojas de la parte frontal de las unidades de disco. Un LED emite luz cuando se aplica un voltaje en la dirección positiva. En el optoacoplador el LED no está visible externamente, pero, en cambio, ilumina un fototransistor que permite que la corriente pase a través suyo cuando cae luz sobre él. La señal lógica (*low* o *high*) de la entrada se transfiere entonces a la salida sin ninguna conexión eléctrica. La MIDI especifica que el optoacoplador ha de necesitar una corriente de menos de 5 mA para encender el LED, y que los tiempos de ascenso y caída en la salida deben ser inferiores a dos microsegundos

Byte de datos en serie MIDI

Kevin Jones

cuerdas apropiadas. Así, la música se codificaba en el papel como una secuencia de eventos de pulsaciones de teclas.

La distancia de un agujero *a través* del papel representaba la altura de la nota, y la distancia *a lo largo* del papel representaba el momento en el cual había de tocarse la nota. ¡Era concebible, entonces, *editar* la música! Por ejemplo, si se perforaba una nota equivocada, se la podía localizar y tapar, mientras se perforaba en el rollo la nota correcta. La codificación de los datos de este modo tiene muchas analogías directas con los primeros sistemas de almacenamiento de datos en ordenador, que empleaban fichas perforadas.

Un procesador de música moderno se puede contemplar como una versión actualizada del sistema de pianola. El papel se reemplaza por la memoria del ordenador y el soporte de almacenamiento magnético, y el piano se reemplaza por un sintetizador con una interface de ordenador. Ahora la edición es simplemente cuestión de procesar los datos de la memoria del ordenador mediante un programa adecuado.

Pero antes de que podamos hacerlo (y, en realidad, incluso antes de que podamos introducir los datos en el ordenador) necesitamos saber dos cosas fundamentales acerca de la MIDI. La primera de éstas es el tipo de transmisión de datos utilizado y, la segunda, el formato de los datos. Estos dos factores corresponden a los componentes de hardware y software de las especificaciones de la interface; en este capítulo nos ocuparemos del hardware que se requiere para nuestro proyecto.

Transmisión de datos

Lo primero que necesitamos saber es cómo se transmite un byte de datos a través del enlace. La MIDI opera como una interface en serie asíncrona operando a 31.25 Kbaudios. Ello significa que sólo se puede transmitir un bit de datos a la vez, y que el receptor y el transmisor no están sincronizados entre sí mediante señales de reloj comunes. La desventaja de este método de transmisión de datos es su escasa velocidad. No obstante, puesto que el enlace se compone de apenas dos cables, exige cables y conectores muy sencillos y económicos.

La MIDI especifica que se han de utilizar los conectores estándares DIN de cinco patillas y 180° (con conectores profesionales de audio XLR de tres patillas como opción). Esto contribuye a mantener reducido el costo de la interface, favoreciendo su incorporación a instrumentos de precio económico.

La principal crítica de que es objeto la MIDI es, en realidad, su baja velocidad de transmisión en serie y, como veremos, las demoras de transmisión pueden volverse bastante notorias en un sistema grande. Sin embargo, en el lado positivo hay que destacar que la velocidad de transmisión es suficientemente lenta para la limitada velocidad de proceso que pueden afrontar los ordenadores personales.

Los datos MIDI se envían en grupos de 10 bits, cada uno de los cuales representa un byte de datos más un bit de comienzo y un bit de final. Cuando no se están enviando datos, la línea normalmente está establecida a un voltaje alto (+5 V para la MIDI). El inicio de una transmisión se señala mediante la puesta a *low* (0 V) de la línea durante un período de un bit (el bit de comienzo). Entonces se envían los ocho bits de datos (el MSB, bit más significativo, en primer lugar) seguidos por un período de un bit a un nivel más alto señalando el fin de los datos. Por consiguiente, la transmisión de cada byte lleva un total de 10 períodos de un bit, lo que equivale a 10/31.25 K=320 microsegundos. Como podremos comprobar más adelante en el proyecto, éste es un número muy importante que los aspirantes a programadores de MIDI deben tener presente.

Cuando la interface está inactiva, la línea se halla a un nivel de voltaje alto. El receptor inspecciona la línea continuamente atento a la primera aparición de un nivel bajo: el bit de comienzo de una transmisión. El receptor sabe entonces que se está enviando un byte de datos y puede establecer su contador, que cuenta períodos de un bit y medio desde la detección del bit de comienzo. Transcurrido este período, el receptor estará a medio camino del período de bit que contiene el MSB de los datos y, por tanto, puede detectar de forma fiable si el bit es un uno o un cero. Los bits restantes se reciben contando sucesivamente siete períodos de un bit y detectando cada vez el nivel.

Por último, se verifica el bit de detención en busca de un nivel alto, para comprobar que los datos estaban correctamente "delimitados" por los bits de comienzo y de final. Aplicando este método de comunicación, no es necesario que los relojes del receptor y el transmisor estén sincronizados con exactitud, simplificando aún más la interface. Sin embargo, la diferencia no debe ser tan acusada como para causar errores de temporización hacia el fin del byte y, por consiguiente, la MIDI especifica una tolerancia de un 1 % en la frecuencia estándar del reloj.

La interface es relativamente rápida, considerando su naturaleza en serie (la RS232 opera a una velocidad máxima de 19.2 Kbaudios) y, por tanto, los cables de conexión deben tener una longitud máxima de 15 m para evitar tiempos de ascenso y caída excesivamente lentos.

Demoras de transmisión

En este punto conviene analizar la limitación fundamental de la MIDI, a saber, las demoras de transmisión entre instrumentos. En general, un mensaje de nota encendida/apagada se compone de un total de tres bytes. El tiempo que consume enviar este mensaje es, en consecuencia, de 3×320 microsegundos=0.96 ms. Consideremos ahora la situación de un sistema de secuenciador de canales múltiples donde, pongamos por caso, puede ser necesario enviar 20 de estos mensajes a la vez. Esto significa que la última nota del grupo llega aproximadamente 20 ms más tarde de lo previsto.

Muchos secuenciadores que emplean grabación en tiempo real utilizan una resolución de temporización de tres o cuatro milisegundos para reproducir adecuadamente los matices de temporización de una actuación humana. Por tanto, una demora de 20 ms puede conducir a una temporización perceptiblemente deteriorada cuando se reproduce la grabación.

En el próximo capítulo analizaremos el software de las especificaciones MIDI.

El Nuevo Mundo (II)

He aquí la segunda parte del listado completo de nuestro juego de simulación mercantil

El juego está escrito en un BASIC mínimo y, por tanto, se podrá ejecutar en el BBC Micro con unas pocas alteraciones. Las principales modificaciones que es necesario introducir en el listado que ofrecemos atañen a la instrucción para limpiar la pantalla y al método de efectuar una pausa para la pulsación de una tecla. A lo largo del listado, allí donde aparezca la instrucción PRINT CHR$(147), los usuarios del BBC Micro han de reemplazarla por CLS.

El código utilizado para aguardar una pulsación de tecla en la versión de *El Nuevo Mundo* para el Commodore 64 (que es la versión que listamos aquí) implica establecer un bucle para tomar (GET) caracteres desde el teclado. El bucle termina cuando se recibe un carácter distinto a la serie nula. Este tipo de construcción aparece en muchos lugares del listado del programa, del siguiente modo:

<n.º línea>GET I$:IF I$="" THEN<n.º línea>

El BASIC BBC posee una instrucción especial con el mismo efecto que la anterior, pero más simple. Siempre que aparezca esta construcción, los usuarios del BBC deben reemplazarla por:

<n.º línea>I$=GET$

En el BBC Micro, las diferentes modalidades de visualización tienen diferentes requerimientos de memoria. Al objeto de dejar libre para el programa la máxima cantidad de memoria de BASIC, los usuarios del BBC Micro deben seleccionar la modalidad 7. Ésta es la modalidad que se selecciona automáticamente cuando se enciende el ordenador. Sin embargo, para asegurarse por completo, los usuarios de esta máquina quizá deseen añadir una línea extra al comienzo del programa:

```
5 MODE 7
```

```
5300 REM INFORME DE FINAL DE SEMANA
5305 PRINT CHR$(147)
5310 S$="                    DIARIO DE NAVEGACION*":GOSUB 9100
5312 S$="                      ------ -- ----------*      ":GOSUB 9100
5314 GOSUB 9200
5316 PRINT:PRINT"                    FIN DE LA SEMANA";WK
5318 GOSUB 9200.
```

Este bucle comprueba si durante la semana ha fallecido algún tripulante, y reduce consiguientemente el número de tripulantes, CN.

```
5320 X=0
5325 FOR T=1 TO 16
5330 IF TS(T,2)<>-999 THEN 5350
5335 X=X+1
5340 TS(T,1)=0:TS(T,2)=0
5345 CN=CN-1
5350 NEXT
5355 IF X=0 THEN 5400
5358 PRINT:PRINT
5360 S$="DURANTE LA SEMANA PASADA*":GOSUB 9100
```

Si ha muerto toda la tripulación (lo que se indicaría mediante CN=0), el juego termina

```
5365 IF CN>0 THEN 5390
5367 S$="MURIO TODA LA TRIPULACION QUE QUEDABA*":GOSUB 9100
5369 GOSUB 9200:PRINT:GOSUB 9200
5373 S$="EL BARCO VA IRREMISIBLEMENTE A LA DERIVA*":GOSUB 9100
5374 GOSUB 9200
5376 S$="POR EL OCEANO DESIERTO.....*":GOSUB 9100
5378 GOSUB 9200
5380 PRINT:PRINT
5382 S$="          EL JUEGO HA TERMINADO*":GOSUB 9100
5384 PRINT:PRINT
5386 GOSUB 9200:END
5388 GOTO 5386
5390 PRINT X;
5392 IF X=1 TEN S$= "MIEMBRO DE LA TRIPULACION HA FALLECIDO*"
5394 IF X<>1 THEN S$= "MIEMBROS DE LA TRIPULACION HAN FALLECIDO*"
5396 GOSUB 9100
5398 GOSUB 9200
5400 PRINT:PRINT
```

¿Se ha agotado alguna provisión? De ser así, se imprime un mensaje en tal sentido

```
5405 S$="AHORA TE HAS QUEDADO SIN "
5410 FOR T=1 TO 4
5415 IF PA(T)<>-999 THEN 5440
5420 PRINT S$;PS(T)
5425 PA(T)=0
5428 S$="                    Y"
5440 NEXT
5450 GOSUB 9200
5455 REM CALCULAR NUEVA JL
```

Se calcula el total de la fortaleza de la tripulación explorando TS(,) y, si resulta inferior a la mitad de la fortaleza total, se añade tiempo extra

```
5460 X=0
5465 FOR T=1 TO 16
5470 X=X+TS(T,2)
5475 NEXT
5480 IF X>799 THEN 5494
5481 PRINT:PRINT
5482 S$="LA TRIPULACION SE HALLA POR DEBAJO DE LA FORTALEZA
     TOTAL*":GOSUB 9100
5483 S$="DE MANERA QUE EL VIAJE PODRIA DURAR MAS*":GOSUB 9100
5489 EW=EW+((800-X)/800)
5490 JL=JL+INT(EW)
5492 PRINT
5494 S$=K$:GOSUB 9100
5496 GET I$:IF I$="" THEN 5496
5499 RETURN
```

Para generar eventos imprevisibles, se selecciona un número aleatorio y se utiliza para llamar una rutina mediante ON...GOTO

```
5500 REM GENERADOR DE EVENTOS AL AZAR
5502 IF RC=RM THEN RETURN
5503 REM SALIR SI TODOS LOS EVENTOS YA PRODUCIDOS
5504 PRINT CHR$(147)
5505 GOSUB 9200
5510 X=INT(RND(1)*RM)+1
5515 REM GENERAR NUM AL AZAR ENTRE 1 Y RM
5520 IF RR(X)=1 THEN RETURN
5522 REM RETURN SI ESTE EVENTO YA SE PRODUJO
5523 RR(X)=1:RC=RC+1
5524 REM ESTABLECER INDIC. PARA SEÑALAR QUE ESTE EVENTO YA SE
     PRODUJO E INCREMENTAR CONTADOR EVENTOS
5525 ON X GOTO 5540,5570,5570,5570,5570,5600,5700,5800,5850,5900,
     5950,6000,6050
5528 REM IR AL CODIGO APROPIADO PARA ESTE EVENTO
5530 PRINT:S$=K$:GOSUB 9100
5535 GET I$=IF I$="" THEN 5535
5539 RETURN:REM RETORNAR AL BUCLE PRINCIPAL VIAJE
```

Algunos eventos son nefastos...

```
5540 REM EVENTO 1-HOMBRE AL AGUA!
5542 PRINT
5543 S$=" DURANTE LA SEMANA*":GOSUB 9100
5545 PRINT:GOSUB 9200
5546 S$=" 1 PERSONA CAYO POR LA BORDA*":GOSUB 9100
5548 S$=" DURANTE UNA TORMENTA*":GOSUB 9100
5550 PRINT:GOSUB 9200
5552 S$="TU TRIPULACION SE HA REDUCIDO AHORA A*":GOSUB 9100
5554 PRINT CN-1;"MIEMBROS"
5558 FOR T=1 TO 16
5559 REM BUSCAR QUE TRIPULANTE SE PERDIO
5560 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 5566
5562 TS(T,2)=-999:REM MUERTO
5564 T=16
5566 NEXT
5568 GOTO 5530
5570 REM EVENTOS DEL 2 AL 5 - PERDIDA DE PROVISIONES
5572 X=X+1:REM AHORA X INDICA LA PROVISION(1-4)
5574 IF PA(X)=0 OR PA(X)=-999 THEN 5530
5576 REM NINGUNA ACCION SI YA SE HA AGOTADO ESTA PROVISION
5578 PRINT
5580 S$=" DURANTE LA SEMANA*":GOSUB 9100
5582 PRINT:GOSUB 9200
5584 PRINT" PARTE DE TU ";P$(X)
5586 S$=" FUE ARRASTRADA POR LA BORDA*":GOSUB 9100
5588 PRINT:GOSUB 9200
5590 S$="AHORA TIENES APROXIMADAMENTE*":GOSUB 9100
5592 PA(X)=PA(X)-INT(PA(X)/(INT(RND(1)*3)+2))
5593 REM REDUCIR CANTIDAD PROV EN 1/2 1/3 O 1/4
5594 PRINT INT(PA(X)/(CN*PN(X)));
5595 PRINT "Y TE QUEDA PARA ";P$(X);"SEMANAS APROXIMADAMENTE"
5599 GOTO 5530
```

...pero otros son beneficiosos. Por ejemplo, se puede sustituir con pescado la ración de carne

```
5600 REM EVENTO 6 - CAPTURAR PECES
5605 X=0
5610 FOR T=1 TO 16
5615 IF TS(T,2)=-999 THEN X=X+1
5620 REM CONTAR FALLECIDOS ESTA SEMANA
5625 NEXT
5630 IF CN-X<1 THEN RETURN
5635 REM NINGUNA ACCION SI TODA TRIPULACION MUERTA
5640 PRINT
5645 S$=" DURANTE LA SEMANA*":GOSUB 9100
5646 PRINT:GOSUB 9200
5650 X$="UNO DE LOS TRIPULANTES CAPTURO*":GOSUB 9100
5655 X=INT(RND(1)*10)+11
5660 REM ENTRE 10 y 20 KILOS
5662 PRINT X;"KILOS DE PESCADO"
5665 PRINT:GOSUB 9200
5670 S$="AHORA TU PROVISION DE CARNE ES*":GOSUB 9100
5675 S$="SUFICIENTE PARA APROXIMADAMENTE*":GOSUB 9100
5678 IF PA(3)=-999 THEN PA(3)=0
5680 PA(3)=PA(3)+X
5685 PRINT INT(PA(3)/(CN*PN(3)));"SEMANAS"
5690 GOTO 5530
```

Si se estuviera acabando el agua, esta rutina permitirá el reabastecimiento. En la línea 5735 se genera al azar un número entre 10 y 20, que representa los barriles de agua que se añadirán al stock retenido en PA(4)

```
5700 REM EVENTO 7 - RECOGER AGUA
5705 PRINT
5710 S$=" DURANTE LA SEMANA*":GOSUB 9100
5715 PRINT:GOSUB 9200
5720 S$="UN TEMPORAL DE LLUVIA LLENO TUS*":GOSUB 9100
5725 S$="BARRILES DE AGUA*":GOSUB 9100
5730 PRINT:GOSUB 9200
5735 X=INT(RND(1)*10)+11
5736 REM ENTRE 10 y 20 BARRILES
5740 S$="AHORA TU PROVISION DE AGUA ES*":GOSUB 9100
5745 S$="SUFICIENTE PARA APROXIMADAMENTE*":GOSUB 9100
5748 IF PA(4)=-999 THEN PA(4)=0
5750 PA(4)=PA(4)+X
5755 PRINT INT(PA(4)/(CN*PN(4)));"SEMANAS"
5760 GOTO 5530
5800 REM EVENTO 8 - VIENTOS FAVORABLES
5805 PRINT
5810 S$="FUERTES VIENTOS CONTINUOS DURANTE TODA LA
     SEMANA*":GOSUB 9100
5815 PRINT:GOSUB 9200
5820 S$="HAS NAVEGADO A BUENA VELOCIDAD*":GOSUB 9100
5825 S$="Y LA DURACION DEL VIAJE SE *":GOSUB 9100
5830 S$="REDUCE EN MEDIA SEMANA*":GOSUB 9100
5835 EW=EW-.5
5839 GOTO 5530
```

El buen tiempo puede mejorar la fortaleza de la tripulación. Esto se consigue explorando la matriz (TS(.,)) e incrementando los coeficientes individuales en función de un número entre 5 y 15

```
5850 REM EVENTO 9 - BUEN TIEMPO
5855 PRINT
5860 S$="BUEN TIEMPO DURANTE TODA LA SEMANA*":GOSUB 9100
5865 PRINT:GOSUB 9200
5870 S$="LA TRIPULACION SE SIENTE MAS FELIZ*":GOSUB 9100
5875 GOSUB 9200
5880 S$="        Y ESTA MAS SALUDABLE!*":GOSUB 9100
5882 FOR T=1 TO 16
5884 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 5888
5886 TS(T,2)=TS(T,2)+INT(RND(1)*11)+5
5888 NEXT
5889 GOTO 5530
```

Las tres rutinas siguientes pueden, al ser llamadas, disminuir el cargamento al estropearse las armas, las balas de tela y los frascos

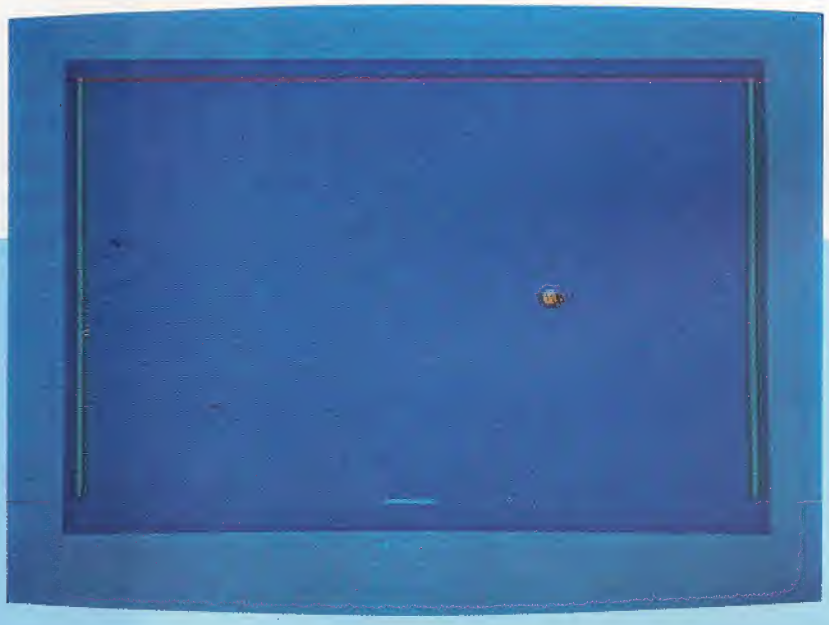
```
5900 REM EVENTO 10 - PERDIDA DE MEDICINAS
5905 IF OA(1)=0 OR OA(1)=-999 THEN RETURN
5910 PRINT
5915 S$="DESCUBRES QUE LA MITAD DE TUS*":GOSUB 9100
5920 S$="FRASCOS DE MEDICINA SE HAN ROTO*":GOSUB 9100
5925 OA(1)=INT(OA(1)/2)
5930 PRINT:GOSUB 9200
5935 S$="AHORA TE QUEDAN*":GOSUB 9100
5940 PRINT OA(1);"FRASCOS SOLAMENTE"
5945 GOTO 5530
5950 REM EVENTO 11 - ARMAS OXIDADAS
5955 IF OA(2)=0 OR OA(2)=-999 THEN RETURN
5960 PRINT
5965 S$="DESCUBRES QUE LA MITAD DE TUS*":GOSUB 9100
5970 S$="ARMAS SE HAN OXIDADO*":GOSUB 9100
5972 S$="Y YA NO SIRVEN*":GOSUB 9100
5975 OA(2)=INT(OA(2)/2)
5980 PRINT:GOSUB 9200
5985 S$="AHORA TE QUEDAN*":GOSUB 9100
5990 PRINT OA(2);"ARMAS SOLAMENTE"
5995 GOTO 5530
6000 REM EVENTO 12 - PERDIDA DE TELA
6005 IF OA(4)=0 OR OA(4)=-999 THEN RETURN
6010 PRINT
6015 S$="DESCUBRES QUE LOS RATONES*":GOSUB 9100
6020 S$="SE HAN COMIDO LA MITAD DE TUS*":GOSUB 9100
6022 S$="BALAS DE TELA Y AHORA*":GOSUB 9100
6024 S$="ESTAS YA NO SIRVEN*":GOSUB 9100
6025 OA(4)=INT(OA(4)/2)
6030 PRINT:GOSUB 9200
6035 S$="AHORA TE QUEDAN*":GOSUB 9100
6040 PRINT OA(4);"BALAS SOLAMENTE"
6045 GOTO 5530
6050 REM EVENTO 13 - ALBATROS
6055 PRINT:A$="S"
6060 S$="UN ALBATROS SOBREVUELA EL BARCO*":GOSUB 9100
6062 GOSUB 9200
6065 S$="ESTE ES UN BUEN PRESAGIO*":GOSUB 9100
6068 S$="Y LA TRIPULACION ESTA CONTENTA*":GOSUB 9100
6070 PRINT:GOSUB 9200
6075 IF PA(3)<(CN*PN(3)*(JL-WK+1)) THEN 6090
6080 REM NO HAY ESCASEZ DE CARNE
6085 GOTO 6122
6090 S$="SE TE ESTA TERMINANDO LA CARNE*":GOSUB 9100
6095 S$="Y EL AVE PESA 10 KILOS!*":GOSUB 9100
6100 PRINT:GOSUB 9200
6105 S$="TE GUSTARIA COGERLA?*":GOSUB 9100
6110 INPUT I$
6112 PRINT:GOSUB 9200
6115 IF LEFT$(I$,1)="S" THEN 6133
6120 S$="PROBABLEMENTE DA LO MISMO!*":GOSUB 9100
6122 PRINT:GOSUB 9200
6125 S$="EL ALBATROS SE ALEJA VOLANDO.....*":GOSUB 9100
6130 GOTO 5530
6133 IF OA(2)=0 OR OA(2)=-999 THEN 6180
6135 S$="HACES UN DISPARO.....*":GOSUB 9100
6138 GOSUB 9200:GOSUB 9200
6140 IF RND(1)<.5 THEN 6150
6145 S$=".....PERO NO ACIERTAS!*":GOSUB 9100
6148 GOTO 6122
6150 S$="Y EL AVE CAE SOBRE LA CUBIERTA!*":GOSUB 9100
6155 PRINT:GOSUB 9200
6160 IF PA(3)=-999 THEN PA(3)=0
6162 PA(3)=PA(3)+10:B$="S"
6165 S$="AHORA TIENES 10 KILOS MAS*":GOSUB 9100
6167 S$="DE CARNE .....*":GOSUB 9100
6170 S$="PERO PUEDE SER QUE LA BUENA FORTUNA*":GOSUB 9100
6172 S$="NO TE ACOMPAÑE A PARTIR DE AHORA!!*":GOSUB 9100
6174 GOTO 5530
6180 S$="NO PUEDES - NO TIENES ARMAS*":GOSUB 9100
6190 GOTO 6122
```

Los eventos ya reseñados poseen una mínima influencia sobre el viaje, pero también se pueden producir otros más determinantes, seleccionados al azar mediante esta rutina

```
6500 REM EVENTOS MAYORES
6508 X=INT(RND(1)*10)+1
6510 ON X GOSUB 6530,6700,6800,6900,7000,7050
6520 RETURN
```

Squash en el C64

Un juego deportivo no hace mal a nadie. Éste le ofrece la posibilidad de practicarlo si es usuario de un Commodore 64



La raqueta se desplaza con ayuda de las teclas de control del cursor (a la derecha de la barra espaciadora). Dispone de diez pelotas que ha de mantener en juego el mayor tiempo posible. Cada pelota que se devuelve proporciona un punto.

```
  5 REM ************
 10 REM *  SQUASH  *
 15 REM ************
 20 GOTO 1000
 50 V=V+DV
 60 H=H+DH
 70 POKE M+B,BN
 80 B=V*X+H
 90 POKE M+B,BO
100 POKE N+B,C
110 IF V=V1 AND ABS(H-(POS(X)+3))>1 THEN 2000
120 IF V=V1 OR V=V0 THEN DV=-DV
130 IF H>=H1 OR H<=H0 THEN DH=-DH
140 IF V=V1 THEN S=S+1
150 GET X$
160 IF X$=C$ AND POS(X)<>0 THEN PRINT GG$;:IF V=V1
    THEN H=H-1
170 IF X$=D$ AND POS(X)<>34 THEN PRINT DD$;:IF V=V1
    THEN H=H+1
180 PRINT R$;
190 PRINT T$;
200 GOTO 50
1000 X=40
1010 M=1024
1020 N=55296
1030 D$=CHR$(29)
1040 G$=CHR$(17)
1050 R0=0
1060 R1=33
1070 BN=32
1080 B=0
1090 C=8
1100 B0=81
1110 H=INT(RND(TI)*(X-4)+2)
1120 V=22
1130 H0=2
1140 H1=X-2
1150 V0=1
1160 V1=22
1170 DV=-1
1180 DH=(RND(TI)<0.5)*2+1
1190 R$=CHR$(32)+CHR$(32)+CHR$(196)
     +CHR$(196)+CHR$(196)+CHR$(32)+CHR$(32)
1200 R=17
1220 B=V*X+H
1230 IF NB=0 THEN GOSUB 1500
1240 GOTO 50
1500 PRINT CHR$(147);
1510 FOR I=0 TO 40
1520 POKE M+I,100
1530 POKE N+I,2
1540 NEXT I
1550 FOR I=1 TO 22
1560 POKE M+I*40,103
1570 POKE N+I*40,5
1580 POKE M+I*40+39,101
1590 POKE N+I*40+39,5
1600 NEXT I
1610 DD$=CHR$(29)+CHR$(29)
1620 GG$=CHR$(157)+CHR$(157)
1630 T$=""
1640 FOR I=1 TO 7
1650 T$=T$+CHR$(157)
1660 NEXT I
1670 PRINT CHR$(19)
1680 FOR I=1 TO 22
1690 PRINT G$;
1700 NEXT I
1710 FOR I=1 TO 16
1720 PRINT CHR$(32);
1730 NEXT I
1740 S=0
2000 NB=NB+1
2010 POKE M+B,BN
2020 POKE 54296,15
2030 POKE 54277,190
2040 POKE 57278,136
2050 POKE 54273,17
2060 POKE 54272,37
2070 POKE 54276,65
2080 FOR I=1 TO 50
2090 NEXT I
2100 POKE 54272,0
2110 POKE 54273,0
2120 POKE 54276,0
2130 FOR I=1 TO 500
2140 NEXT I
2150 IF NB=11 THEN 3000
2160 GOTO 1000
3000 IF S>RR THEN RR=S
3010 NB=0
3020 PRINT CHR$(147);
3030 FOR I=1 TO 7
3040 PRINT
3050 NEXT I
3060 PRINT TAB(16)"PUNTOS[1SPC]:";
3070 PRINT S
3080 FOR I=1 TO 4
3090 PRINT
3100 NEXT I
3110 PRINT TAB(12)"PUNTUACION[1SPC]MAXI
     MA[1SPC]:";
3120 PRINT RR
3130 FOR I=1 TO 4
3140 PRINT
3150 GET X$
3155 GET X$
3160 NEXT I
3170 PRINT TAB(16)"OTRA[1SPC]?"
3180 GET X$
3190 IF X$="" THEN 3180
3200 IF X$<>"N" THEN 20
3210 END
```

Centro musical

Paul Chavo

Teclado avanzado
El teclado Echo, de LVL, que vemos en la ilustración, es el primer teclado musical "verdadero" producido para el BBC Micro y el Commodore 64 que accede directamente a los chips de sonido de estos ordenadores y que no exige una costosa interface

El paquete de música Echo, destinado al Commodore 64 y al BBC Micro, accede directamente al chip de sonido del micro

El paquete Echo, de Leasalink Viewdata Ltd (LVL), consta de un auténtico teclado de tres octavas y media más el software para hacerlo funcionar. También se proporciona un manual y un cable interface de 20 vías para conectar el teclado y el ordenador entre ellos mismos, que se instala en la puerta para el usuario de la máquina. Si el usuario posee un Commodore 64, se le entregarán dos cables: el cable estándar, que es para el BBC Micro, y un pequeño cable adaptador que se ofrece para producir la interface correcta para la máquina de Commodore.

El hardware del Echo es simple desde el punto de vista de diseño; la mayor parte del trabajo lo realiza el software que viene con el paquete que, obviamente, mantiene reducidos los costos de producción. El software para ambos ordenadores se basa en las mismas especificaciones.

En el software hay disponibles dos modalidades distintas, la primera de las cuales es una modalidad de órgano. Pulsando una de las teclas del teclado del ordenador se seleccionará uno de los diversos "instrumentos" listados. Éstos van desde la guitarra hawaiana hasta el cello y el clavicordio, aunque los fondos de cada una de las dos máquinas son ligeramente diferentes. En la parte superior del monitor se listará una cantidad de parámetros que, según cuál seleccione usted, le permitirán alterar los tonos del instrumento que esté tocando.

La versión del software para el Commodore 64 posee la facilidad de añadir trémolo y vibrato (pero no ambos) o cambiar a una clave mayor o menor. Estos efectos se producen pulsando una de las teclas numéricas.

El software de la versión para el BBC Micro también incorpora las teclas del ordenador, pero en este caso son las teclas de función las que le permitirán establecer el trémolo y acordes mayores o menores. Los programadores han sacado partido de las teclas de función extras y de la velocidad del procesador de la máquina, por lo que han añadido otras facilidades. Si el usuario desea dar un énfasis o un efecto extra cuando ejecuta una pieza en el teclado Echo, puede pulsar una de las teclas de función y se tocará el sonido apropiado (bajo, cimbal, etc.). Lamentablemente, no hay ninguna facilidad que permita tocar ritmos de acompañamiento mientras se esté tocando el teclado.

Controles de altura

En la parte inferior de cada pantalla están los controles de altura. En la versión para el BBC Micro, se puede subir o bajar la altura pulsando las teclas del cursor adecuadas. Sin embargo, en la versión para el Commodore 64 esta labor se realiza mediante las teclas < y >. Lamentablemente, el hecho de que estas dos funciones aparezcan en la pantalla etiquetadas como PITCH MINUS no dice mucho en favor de la calidad del software para la máquina Commodore.

También es lamentable que los sonidos producidos por las teclas de "instrumentos" guarden muy poco parecido con los instrumentos que se pretende imitar. Éste es un hecho harto frecuente en todo el campo de la música electrónica, no sólo en los programados para ordenador. No obstante, los efectos de sonido del Echo parecen cualitativamente peores que la mayoría. Por ejemplo, en la versión para el Commodore 64 el violín, la viola y el cello suenan meramente como un órgano sostenido a diferentes alturas para cada instrumento.

Ambas máquinas disponen, asimismo, de la modalidad de sintetizador. Los parámetros proporcionados por la modalidad de sintetizador dependen de las facilidades que ofrece el chip de sonido de cada máquina. De este modo, la versión para el Commodore 64 permite regular los parámetros de envoltura de la forma de onda y programar el filtro. Tal como sucede en la modalidad de órgano, los parámetros se alteran pulsando la tecla adecuada en el teclado del ordenador hasta obtener el valor requerido.

De una forma bastante similar, la modalidad de sintetizador del BBC Micro hace uso de la instruc-

ECHO SYNTHESISER

TECLADO

37 teclas de recorrido total

INTERFACES

Cable de 20 vías que se enchufa en la puerta para el usuario del BBC Micro. Cable adaptador para el C64

SOFTWARE

La versión para el BBC Micro se vende en cassette o en disco; la versión para el C64 sólo en cassette

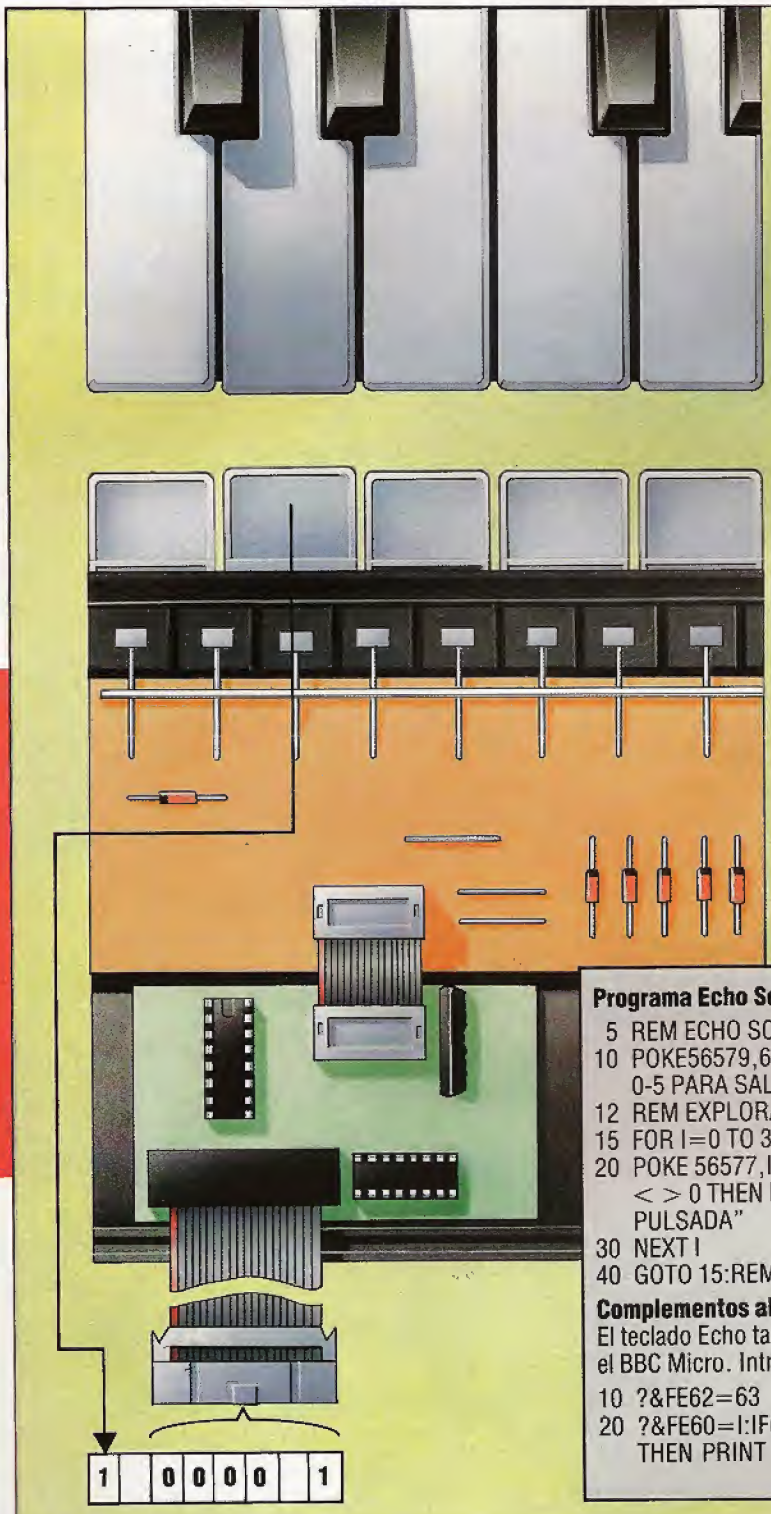
VENTAJAS

Permite utilizar, desde un auténtico teclado, las facilidades de los chips de sonido de ambos micros

DESVENTAJAS

La pobre calidad del software reduce sus posibilidades, si bien existe la promesa de una versión mejorada

Música en pantalla
Éstas son las visualizaciones en pantalla para el software del Commodore 64 (arriba) en modalidad de órgano, y para el BBC Micro en modalidad de sintetizador. A pesar de ser superficialmente parecidas, las versiones del software son bien diferentes. Aunque ambas utilizan pulsaciones de teclas desde el ordenador para alterar los sonidos, la versión para el BBC Micro posee una variedad de fondos muchísimo mayor que la versión para el Commodore 64



Echo Scan

El teclado Echo se enchufa en la puerta para el usuario del BBC Micro o del Commodore 64 y se lo puede integrar fácilmente desde software. El principio de operación es muy simple. Los seis bits inferiores del registro de datos de la puerta para el usuario se establecen en salida para poder enviar datos al teclado, y el bit más significativo (el 7) se establece en entrada para volver a recibir datos desde el teclado. Para comprobar una pulsación de tecla es preciso, en primer lugar, colocar (POKE) un número entre 0 y 36 en el registro de datos de la puerta para el usuario (el teclado Echo posee 37 teclas numeradas a partir de cero desde el extremo izquierdo). Éste define a la tecla que deseamos comprobar. Luego debemos comprobar el bit 7 del registro de datos (operando el valor del registro con 128 mediante AND). Si este bit es uno, entonces en ese momento se está pulsando la tecla en cuestión; si el valor del bit es cero, ello indica que no se está pulsando la tecla. Para explorar todo el teclado necesitamos, por lo tanto, un bucle que compruebe cada una de las 37 teclas de forma sucesiva. El programa que ofrecemos muestra cómo se puede hacer esto desde BASIC.

Programa Echo Scan

```
 5 REM ECHO SCAN CBM 64
10 POKE56579,63:REM ESTABLECER BITS
   0-5 PARA SALIDA
12 REM EXPLORAR TECLADO DE 37 TECLAS
15 FOR I=0 TO 36
20 POKE 56577,I:IF(PEEK(56577)AND 128)
   <> 0 THEN PRINT I; "TECLA
   PULSADA"
30 NEXT I
40 GOTO 15:REM REPETIR
```

Complementos al BASIC

El teclado Echo también se puede utilizar con el BBC Micro. Introduzca estos cambios:

```
10 ?&FE62=63
20 ?&FE60=I:IF(?FE60 AND 128) <> 0
   THEN PRINT I; "TECLA PULSADA"
```

Kevin Jones

ción ENVELOPE de la máquina. Cuando llame a esta instrucción, se le preguntará cuál de los cuatro sintetizadores definidos por el usuario (en contraposición a aquellos que ya están preestablecidos en el órgano) desea alterar. La pantalla visualiza entonces los 14 parámetros que se requieren para la instrucción ENVELOPE; cada uno de ellos se selecciona mediante las teclas del cursor izquierda y derecha, y sus valores se pueden modificar mediante los cursores arriba y abajo. Una vez establecidos, usted estará en libertad de tocar el tono deseado en el teclado.

El software conecta en interface el ordenador con el teclado colocando un valor de entre 36 y cero en los seis bits inferiores del registro de datos de la puerta para el usuario. El método de exploración de teclado que utiliza el paquete de software significa que si se mantienen pulsadas varias teclas simultáneamente, sólo se registrará la más elevada. Lamentablemente, esto no permite cambios rápidos de dedos en el teclado del Echo, porque antes de que el software reconozca una pulsación de tecla, debe haberse liberado completamente la tecla anterior.

Sonidos de circuito

Prosiguiendo con nuestro proyecto, diseñaremos una interface para el Commodore 64 y el BBC Micro

El principal componente de la interface es un adaptador de interface para comunicaciones asincrónicas (ACIA: *Asynchronous communications interface adaptor*). La función del ACIA consiste en convertir los datos en paralelo del ordenador a la forma en serie que requiere la MIDI, y viceversa. El dispositivo se halla contenido en un circuito integrado. Nosotros emplearemos el MC6850, presente en muchos ordenadores personales como un control para la transferencia asíncrona de datos desde y hacia cassette e interfaces de tipo RS232.

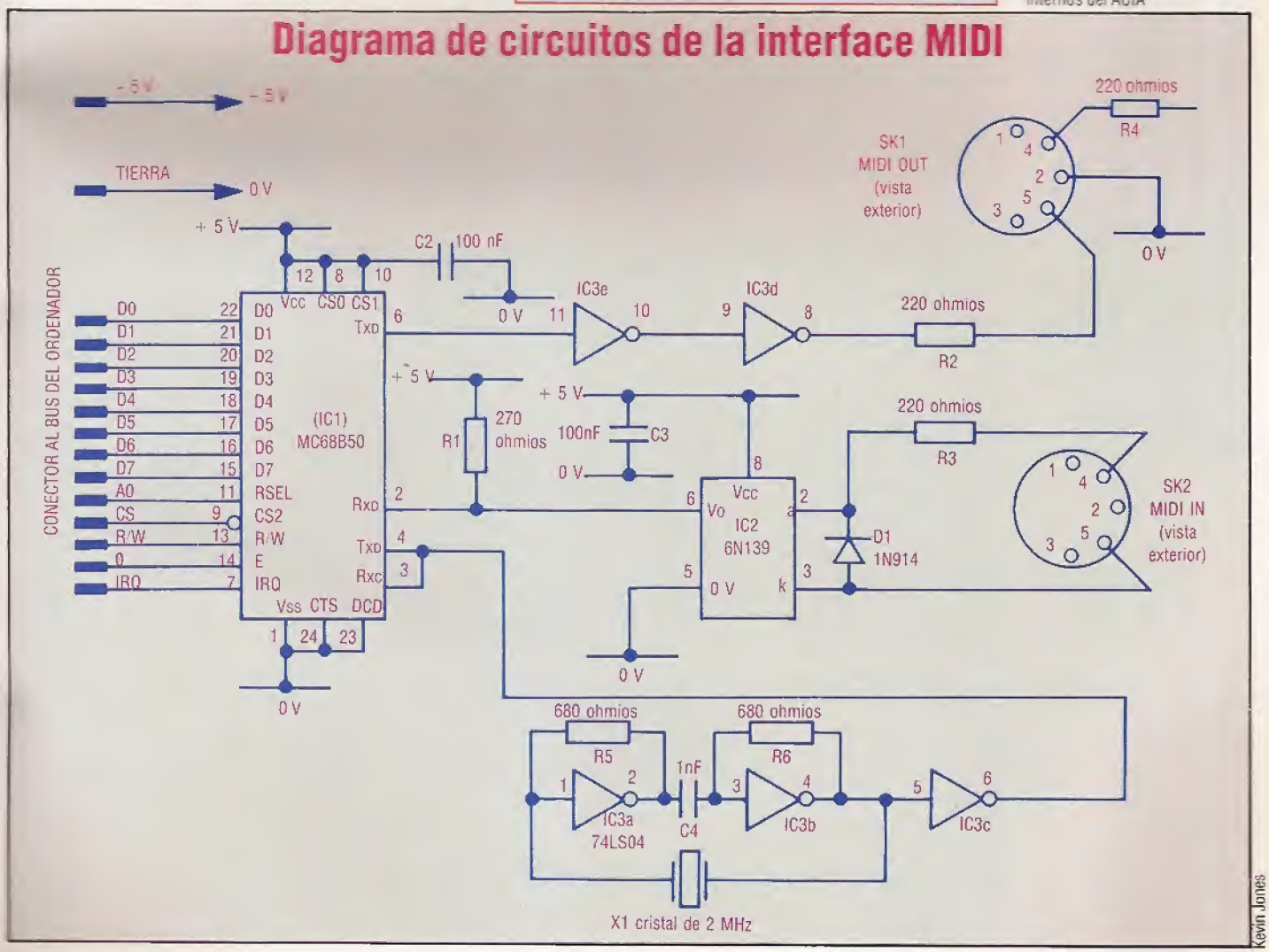
Lista de componentes

Cantidad	Ref.	Artículo
1	C4	Condensador policarbonato 1 nF
2	C2/C3	Condensador policarbonato 100 nF
1	R1	Resistencia 270 ohmios
3	R2/R3/R4	Resistencia 220 ohmios
2	R5/R6	Resistencia 680 ohmios
1		Carrete de alambre para enlaces
2	SK1/SK2	Conector DIN 5 patillas 180° para montaje en PCB
1	IC1	Chip ACIA MC68B50
1	IC2	Chip optoaislante 6N139
1	IC3	Inversor hexa 74LS04 TTL
1		Conector DIL 24 patillas
1		Conector DIL 8 patillas
1		Conector DIL 14 patillas
1	X1	Cristal 2.0 MHz
1	D1	Diodo 1N914

La placa DIP que se utiliza para montar los componentes se puede conseguir en casi todas las sucursales Tandy, bajo el número de componente 276-164

Conexiones del circuito
El optoacoplamiento se consigue con un optoaislante 6N139 (IC2). El reloj de transmisión es derivado del reloj del ordenador mediante un circuito oscilador de cristal de 2MHz formado alrededor de IC3a e IC3b. IC3c, IC3d e IC3e proporcionan cada uno funciones de tamponamiento para el transmisor y generador del reloj. El MC6850 es bus compatible con la CPU 6502 utilizada en muchos ordenadores. Ello nos permite efectuar conexiones directas desde las líneas Read/Write, IRQ y Clock de micros basados en el 6502. (Observe que, para evitar posibles confusiones con las entradas del reloj de transmisión TXD y TXC, hemos etiquetado la entrada del reloj al ACIA como Enable (E)). Las dos líneas de selección del chip (CS0 y CS1) están conectadas a la alimentación de + 5 V y, por tanto, fijadas en *high*, dejando a CS2 como la única entrada activa de selección de chip (o habilitador de dispositivo). Esta línea queda *low* para una cierta gama de direcciones, permitiendo que las operaciones de lectura y escritura en el bus de datos accedan a los registros internos del ACIA

Diagrama de circuitos de la interface MIDI



Kevin Jones

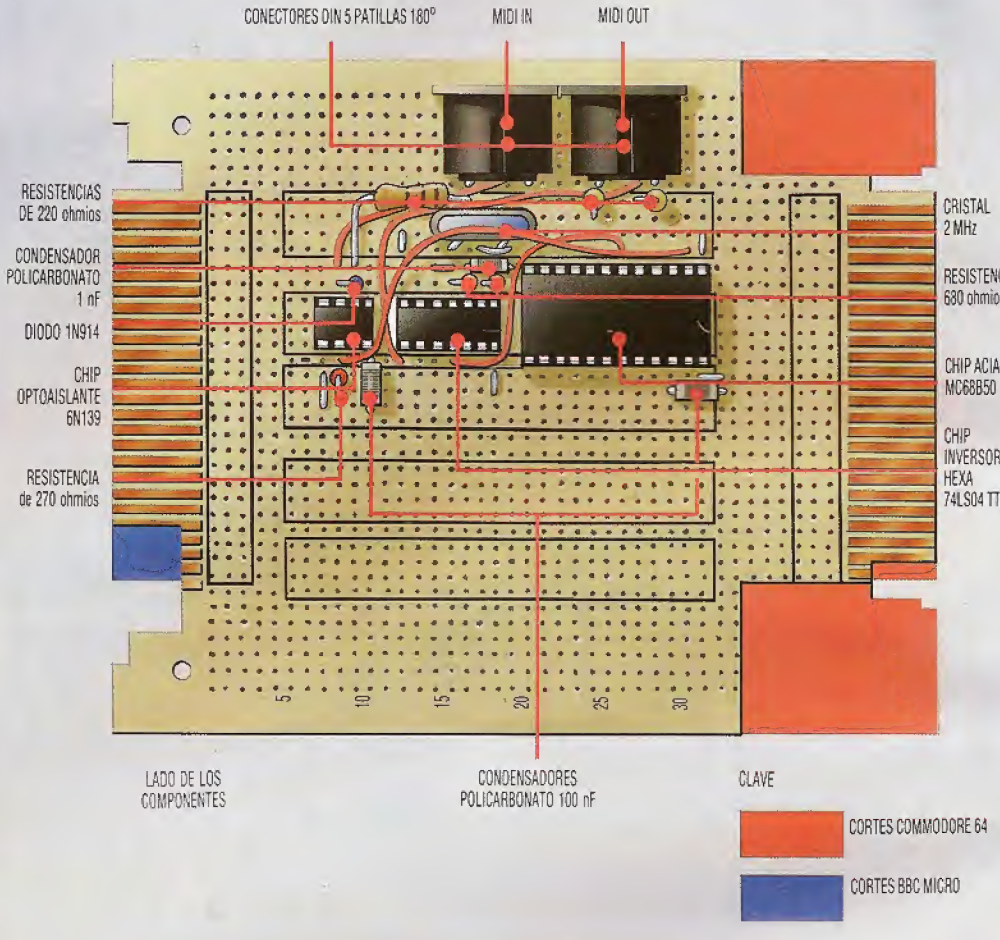
Construcción de la MIDI

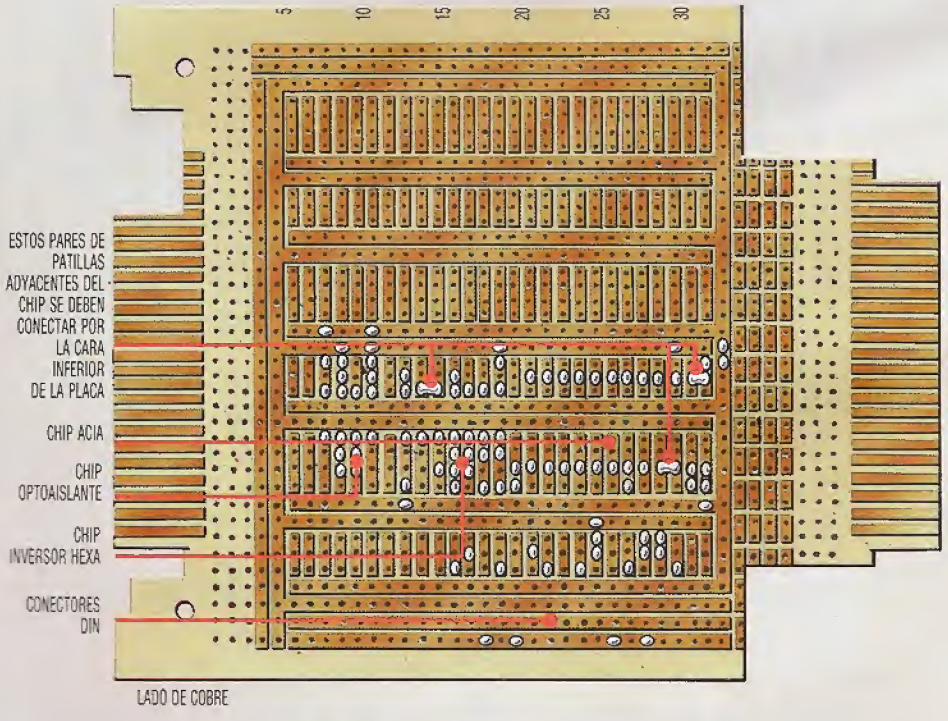
Los componentes principales de la interface MIDI se deben montar en la placa DIP especial que se especifica en la lista de componentes. Esta placa posee conectores marginales de doble cara en ambos extremos; el extremo derecho se utiliza para la conexión al Commodore 64, y el extremo izquierdo para la conexión al BBC Micro. Antes de montar los componentes es preciso efectuar algunos cortes en la placa. Observe que sólo es necesario realizar cortes en un extremo de la placa o bien en el otro, según con qué ordenador desee utilizar la interface. Emplee un cuchillo puntiagudo o una pequeña sierra para metales para suprimir las porciones requeridas, tal como se indica. Todos los enlaces se efectúan en la cara superior de la placa utilizando alambre para enlaces de una sola hebra, con la excepción de tres enlaces entre patillas IC adyacentes. Estos enlaces se han de realizar haciendo correr soldadura entre los pares de patillas apropiados por el lado de cobre de la placa.

Comience por montar en la placa los componentes pasivos: las resistencias, los condensadores, los conectores DIN y los conectores DIL, tal como se indica. Efectúe los enlaces necesarios con el alambre de enlaces y monte el cristal de 2 MHz. El diodo debe orientarse de modo que el extremo marcado con la franja de color quede a la derecha (mirándolo desde arriba). Asegúrese de no fallar el pequeño enlace de debajo del condensador C3. Por último, coloque cuidadosamente los chips en su sitio, en sus respectivos conectores, prestando atención a la orientación de las muescas.

En el próximo capítulo del proyecto explicaremos de forma detallada el trabajo que aún resta por hacer antes de que se pueda conectar la interface a un ordenador y comprobar su funcionamiento

Diagrama de la disposición de los componentes





Otros componentes que necesitaremos en esta etapa son un reloj de interface, un optoaislante y conectores de entrada y salida, que constituyen la base de la interface. En el próximo capítulo reseñaremos de forma detallada las conexiones a la puerta del ordenador que se requieren para enlazar la interface con el Commodore 64 o el BBC Micro.

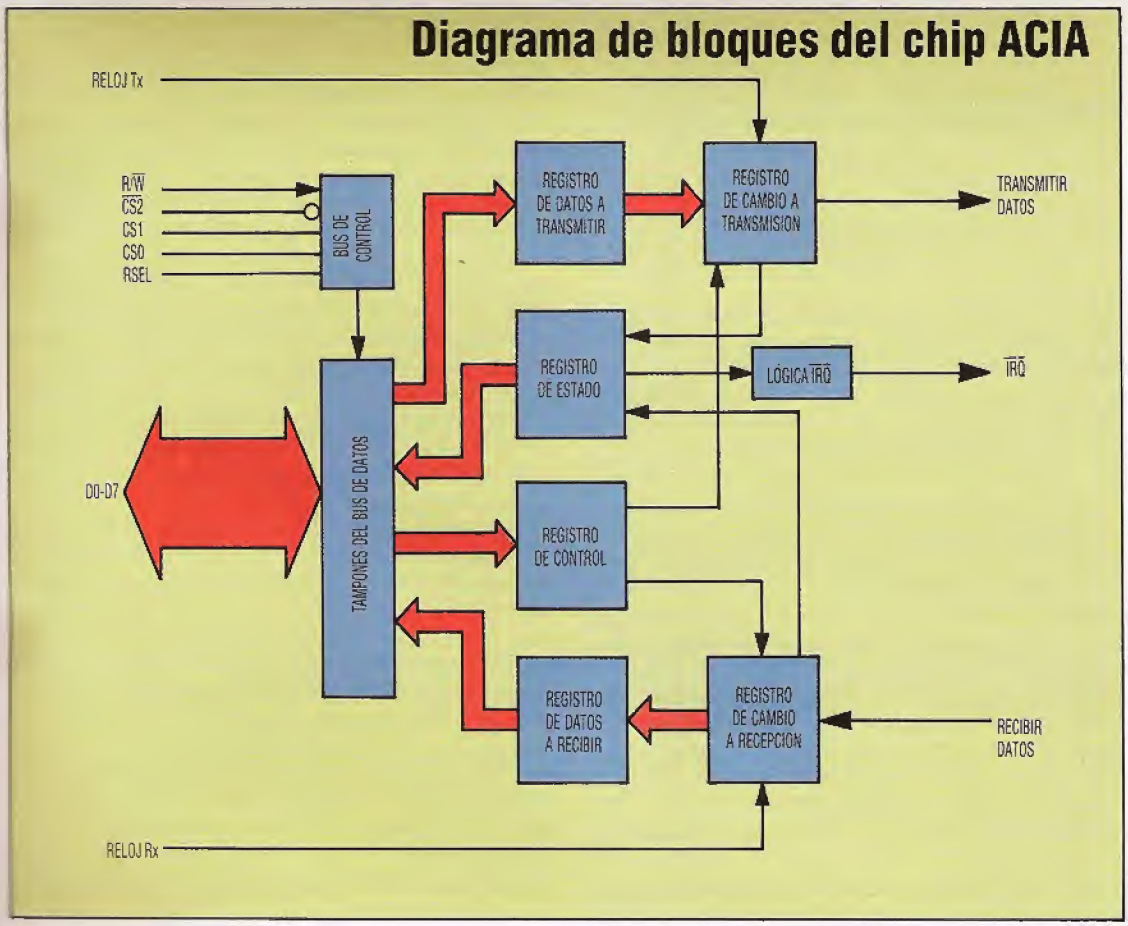
La estructura interna del dispositivo ilustrado se compone de un cierto número de registros de ocho bits. La línea de entrada *Register* SELect (RSEL) permite el acceso a los registros desde el bus de datos. Aunque no es esencial comprender las funciones de los registros del ACIA para ejecutar el software que daremos, los que deseen programar la interface necesitarán conocer la siguiente información:

- El *registro de estado* se compone de ocho bits que describen el estado actual del chip ACIA. El programador los tiene a su disposición llevando a cabo una operación de lectura en el ACIA con la línea RSEL a nivel *low* (cero lógico).
- El *registro de control* contiene ocho bits que controlan la operación del ACIA. Al registro de control se accede escribiendo en el ACIA con la línea RSEL en un nivel *low*.
- El *registro de cambio a transmisión* (TSR: *transmit shift register*) realiza la conversión paralelo a serie que se requiere para transmitir un byte de datos. El registro se carga con un byte desde el *registro de datos a transmitir* (TDR: *transmit data register*) cada vez que el TDR está lleno y se haya completado la transmisión del byte anterior. Esta operación establece el bit 1 del registro de estado. Luego el byte es transportado a una velocidad determinada por el reloj de transmisión. Entonces a los ocho bits de datos se les añaden los bits de comienzo y final. Éste es un registro interno al cual no se puede acceder directamente desde el bus de datos.
- El *registro de datos a transmitir* (TDR) forma un tampón (o *buffer*) entre el TSR y el bus de datos del sistema. El byte a transmitir se carga en este registro escribiendo en el ACIA con la línea de selección de registros *high*. Esto limpia el bit 1 del registro de estado. Para no perder los datos, no se debe cargar el TDR cuando este bit está borrado. Ello se debe a que el byte previo está aún en el TDR esperando la transmisión y se escribiría sobre él.
- El *registro de cambio a recepción* (RSR: *receive shift register*) realiza la conversión de serie a paralelo que se requiere cuando se reciben datos desde la MIDI. Cuando el registro recibe un byte completo, carga los datos en el RDR, poniendo a 1 el bit de estado 0. Si ya estaba establecido a 1 con anterioridad, entonces también se establecerá a 1 el bit de estado 5, indicando que el byte anterior no había sido leído por la CPU y que los datos se perderán.

Si el byte recibido no tuvo la cantidad requerida de bits de comienzo y de final, entonces se establecerá a 1 el bit de estado 4. Esto podría ocurrir si en la línea de entrada en serie hubiera presente algún "ruido" eléctrico. Al igual que el TSR, al RSR no se puede acceder directamente.

- El *registro de datos a recibir* (RDR: *receive data register*) se carga cada vez que se recibe un byte completo desde el RSR. Contiene, en consecuencia, el byte de datos más reciente. A él se accede efectuando una operación de lectura del ACIA con la línea RSEL establecida *high*. Esta operación pone a 0 el bit de estado 0.

Diagrama de bloques del chip ACIA



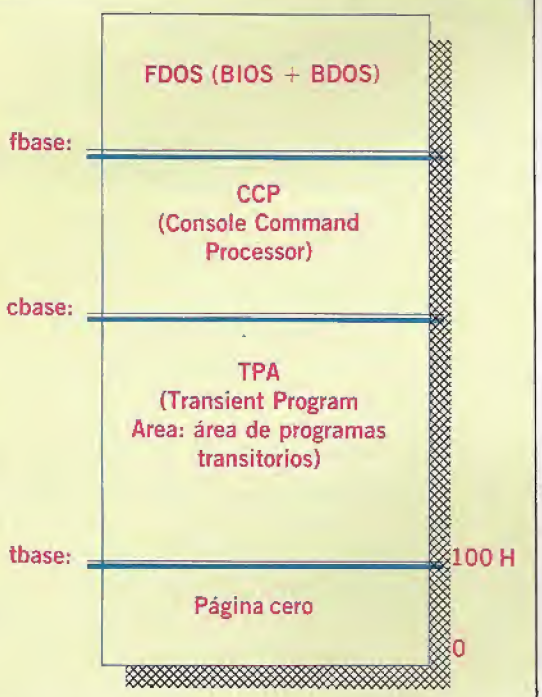
Kevin Jones

Futuro prometedor

En este último capítulo de nuestra serie acerca del CP/M, analizaremos cómo se organiza el sistema en la memoria del ordenador

Un espacio bien aprovechado

Este diagrama ilustra en qué lugar de la memoria se almacenan los diversos componentes del CP/M. En la parte inferior de la memoria, en la página cero, están las variables del sistema y los puntos de entrada, así como el programa cargador básico. Más arriba se halla la TPA (área de programas transitorios). La frontera superior de la TPA se puede alterar para permitir que los archivos que normalmente no cabrían en el área se sobreescriban en el CCP (*console command processor*). En la parte superior de la memoria están las rutinas BIOS y BDOS, que no se pueden sobreescribir porque son importantes para la ejecución de instrucciones



Los tres módulos principales del CP/M se mantienen en la parte superior de la memoria. Sus direcciones de comienzo reales dependerán de la versión de CP/M que se esté utilizando. El BDOS (sistema operativo de disco básico), que administra los archivos retenidos en disco, y el BIOS (sistema básico de entrada/salida), que manipula para el ordenador las rutinas de tratamiento de periféricos, se conservan en la parte más alta de la memoria. Por debajo de ellos está el CCP (*console command processor*), el módulo del CP/M con el cual se comunica el usuario.

En el otro extremo del mapa de memoria hay 256 bytes (la "página cero") que se utilizan para contener variables del sistema y otra información de trabajo necesaria para ejecutar el CP/M. Ésta comprende algunos datos muy útiles, como los puntos de entrada a las zonas BDOS y BIOS de la memoria y el programa cargador básico. Es necesario mantener en la memoria el programa cargador básico porque a menudo el sistema necesitará volver a cargar el CP/M (más adelante explicaremos por qué).

Otra característica importante que está retenida en la página cero es el área conocida como TFCB (*transient file control buffer:* buffer de control de archivos transitorios). En el capítulo anterior descubrimos que cuando el usuario pide que se llame un archivo desde disco, el CCP prepara en la memoria un bloque ficticio de control de archivo. Cuando el BDOS localiza un archivo, lo compara con el archivo retenido por el CCP y después pasa más información desde la pista del directorio al CCP. Esta información está almacenada en el TFCB.

Entre las secciones de la página cero y el CCP hay un área de memoria denominada TPA (*transient program area*: área de programas transitorios). Se trata de la zona que es como el espacio de trabajo del CP/M. Como hemos visto, cuando se somete una instrucción al sistema operativo, el CCP primero busca una pareja en la lista de las instrucciones residentes retenidas en el área CCP. Si no está entre ellas, el CCP da por sentado que la instrucción es transitoria y le ordena al BDOS que la localice. En el supuesto de que la instrucción esté residente en el disco de sistema, el BDOS cargará una copia del programa en código máquina en el principio del área de programas transitorios (TPA), que siempre está en la posición hexa 100 (el comienzo de la página uno en el mapa de memoria del ordenador). Una vez cargada, la instrucción está lista para ser ejecutada, lo que se realizará de forma automática.

Las especificaciones para utilizar el CP/M afirman que se requiere un mínimo de 16 K de RAM. Esta cantidad de memoria es realmente grande, considerando que el CP/M está destinado a permanecer en un segundo plano mientras se estén ejecutando otros programas. Afortunadamente, los programas CP/M propiamente dichos ocupan una pequeña cantidad de esta memoria. La mayor parte de la RAM queda reservada para uso de la TPA, la cual (si usted está utilizando el mínimo) terminará en la dirección hexa 2900. Puesto que en CP/M la página cero está reservada para las variables del sistema, se emplearán 7 K para la TPA.

Sin embargo, la mayoría de las instrucciones transitorias ocupan sólo entre 2 y 3 K de memoria. El resto de la TPA se utilizará para todos los archivos sobre los que necesite trabajar la instrucción transitoria. Por consiguiente, a la TPA sólo la emplean la instrucción que se esté ejecutando y los archivos sobre los cuales se requiere que actúe la misma.

Ello le plantea un problema al sistema. Si tenemos la cantidad mínima de RAM, que son 7 K de espacio de TPA libres, y se utilizan 3 K para la instrucción transitoria, nos quedan apenas 4 K para cualquier archivo extra. Esto apenas si es suficiente para un programa de tamaño moderado, menos aún para archivos de texto extensos. En el capítulo anterior comentamos que un archivo CP/M puede tener hasta 16 K de longitud. Por supuesto, una forma de abordar el problema consiste en añadir memoria extra. Los bancos extras de RAM que se añadan al sistema se le asignarán a la TPA, hasta una cantidad máxima de 64 K, que es la que el CP/M puede direccionar directamente. Por lo tanto, el tope del CP/M se elevará en 4000 hexadecimal por cada 16 K adicionales.

Pero supongamos que no podemos añadir la memoria extra requerida. ¿Cómo consigue el CP/M llevar a cabo esta tarea con menos de la memoria necesaria para ello? La respuesta es, simplemente, que todo sobreflujo proveniente de la TPA se sobreescribe (o *superpone*, en la jerga del CP/M), sobre el CCP. Esto no es tan drástico como parece. Por un lado, cuando se está ejecutando una instrucción no se requiere el CCP. El CP/M, al igual que todos los sistemas operativos, no aceptará ninguna otra instrucción mientras esté ejecutando la anterior. Por consiguiente, no es preciso que esté presente el CCP para interrumpirlas ni para ejecutar instrucciones residentes innecesarias.

Por supuesto, una vez terminado el programa de instrucción, será necesario volver a cargar los programas CCP con el objeto de preparar al sistema para aceptar la instrucción siguiente. Esto se efectúa como el último acto de una instrucción transitoria, y el programa llamará luego a la rutina de *carga de posiciones*, que se encuentra en la posición hexadecimal 0005 de la página cero. Esta dirección es el punto de entrada al sistema operativo que vuelve a cargar en la memoria el módulo CCP, listo para recibir la siguiente instrucción.

Cuando se inventó el CP/M, el hardware disponible (los chips de RAM, en especial) era muy costoso y, por consiguiente, eran pocas las máquinas equipadas con más de 16 K como estándar. De ese modo, todo el software diseñado para ejecutarse en esas máquinas había de hacerse a la medida para adecuarlo a las restricciones del hardware.

Con el fin de lograr que el CP/M fuera un OS lo más amplio posible para el espacio de memoria disponible, hubieron de superponerse algunos componentes. Dado que se requería que tanto el BIOS como el BDOS estuvieran residentes de forma permanente, los mismos no se podían utilizar. Muchas de las instrucciones se valían de ellos para acceder a archivos o efectuar alguna operación de entrada o salida (tales como escribir un archivo por una impresora o visualizarlo en una pantalla). Por tanto, se decidió que el CCP y la TPA "ocuparan" la misma zona de memoria, dado que el uno no se podía utilizar al mismo tiempo que el otro.

La zona CCP podía ser superpuesta por un programa transitorio; una vez ejecutado, éste, se volvía prescindible y el CCP se podía volver a cargar con seguridad.

Parece que ahora que las máquinas de gestión han comenzado a adoptar procesadores de 16 bits, el sistema operativo esté llamado a una inevitable desaparición. Pero aunque los sistemas operativos de 16 bits tales como el MS-DOS dominan en la actualidad el mercado de gestión, parece que el micro de ocho bits basado en el Z80 tiene aún por delante muchos años de utilidad.

Con frecuencia los sistemas de micros personales y los pequeños de gestión no necesitan la potencia de un procesador de 16 bits, pero sus unidades de disco exigen un OS. El CP/M, que durante tanto tiempo ha sido el estándar de facto para sistemas operativos de disco de ocho bits, parece una opción ideal para los fabricantes ansiosos de proporcionar una amplia base de software sin elevar sustancialmente los costos de desarrollo. Algunos fabricantes de ordenadores ya han elegido este camino, en particular Memotech y Amstrad.

Los usuarios del Amstrad tienen una dificultad especial. El CP/M se desarrolló para discos de 8 pulgadas y de 5 1/4 pulgadas, y Amstrad ha optado por el formato Hitachi de 3 pulgadas. Esto significa que habría de transcurrir algún tiempo antes de que aparezcan en el mercado muchos paquetes basados en CP/M para los usuarios del Amstrad.

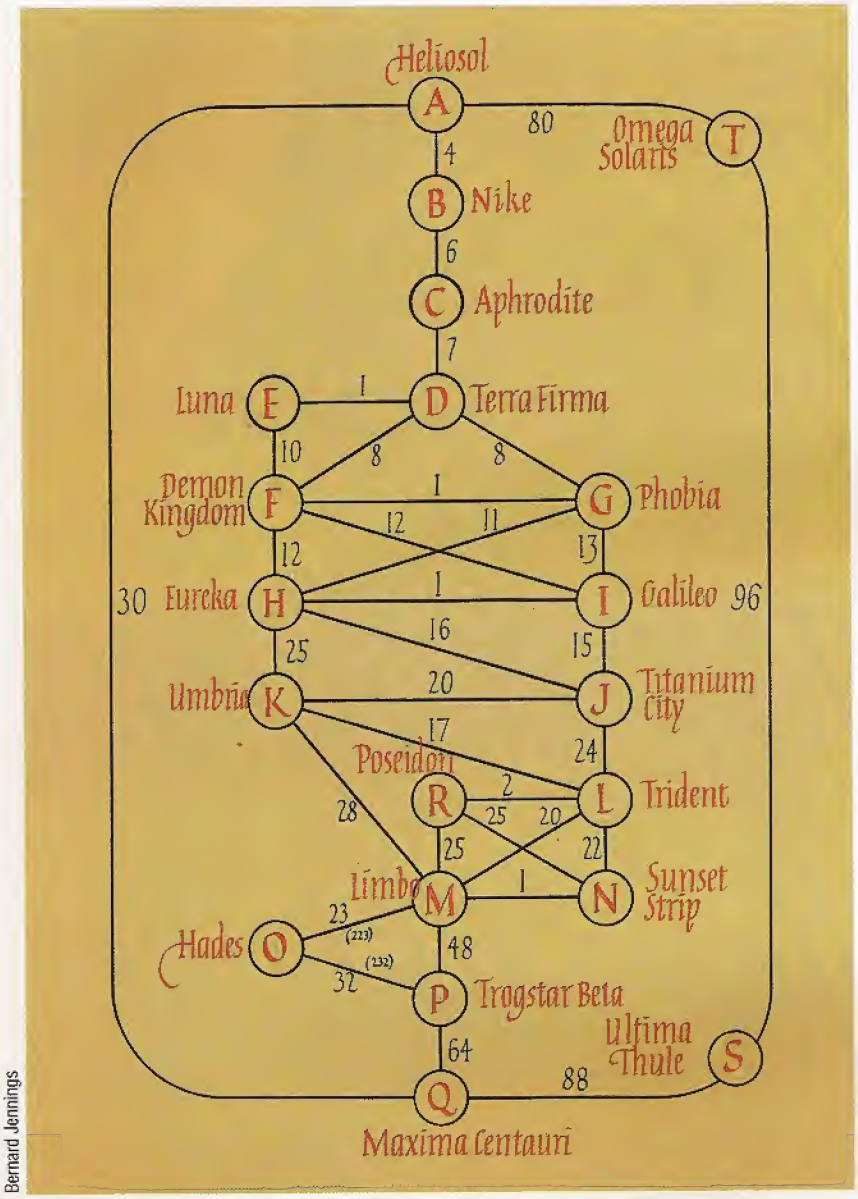
Otro problema del ordenador Amstrad es que el apoyo de su pantalla deja sólo 38 K de memoria libre para programas, que es muchísimo menos de lo que requieren algunos de los programas CP/M más recientes. Por consiguiente, a menos de que algunos de estos programas se puedan adaptar para caber en el espacio de memoria disponible, los usuarios de esta máquina pueden encontrarse con tener que utilizar software anticuado. No obstante, al "ir bajando en el mercado", parece que el CP/M de ocho bits continuará generando una gran base de consumidores.

Esto no equivale a decir que Digital Research haya abandonado su mercado de 16 bits. Pero a la empresa las cosas no le han resultado fáciles, porque ahora hay muchos más competidores de los que había cuando Gary Kildall desarrolló el CP/M. Una posterior versión del sistema, CP/M-86, fue el primer intento de Digital Research por introducirse en el mercado basado en los chips Intel 8088/6. Sin embargo, esta versión experimentó el rechazo general del mercado, que optó por la compatibilidad con IBM y el estándar MS-DOS.

Sin embargo, más recientemente Digital Research ha lanzado una versión multiusuarios del CP/M denominada Concurrent CP/M. Este OS lo puede utilizar un único usuario o bien varios ordenadores enlazados entre sí en forma de red. Quizá sea demasiado pronto para decir si el Concurrent CP/M obtendrá el mismo éxito que su predecesor de ocho bits, pero lo que sí parece seguro es que el CP/M estará en un primer plano, en una forma u otra, durante muchos años.

En el MP/M y el CP/NET

Aquí nos hemos dedicado exclusivamente a la versión 2.2 del CP/M, la versión más común del sistema operativo. Esta versión está diseñada para aplicaciones de un único usuario y una única máquina. Sin embargo, Digital Research ha desarrollado otros (conocidos como "*sistemas multitareas*") basados alrededor del CP/M para utilizar con numerosas máquinas y usuarios. Un sistema multitareas es uno en el cual la CPU comparte su tiempo (en virtud del *fraccionamiento del tiempo*) entre varios usuarios o periféricos. El OS que se ha desarrollado para permitir que numerosas personas utilicen la misma CPU se denomina MP/M (*multi-processing monitor control program*). Utilizando este sistema, un único ordenador se conecta con hasta 16 terminales diferentes, todos los cuales son completamente independientes entre sí. Una persona situada ante un terminal no tendrá conciencia alguna de la existencia de los otros usuarios. Para poder hacer frente a todos estos usuarios, se han incluido en el MP/M varias características diseñadas para ayudar a compartir ya sea los archivos o los dispositivos del sistema como las impresoras. Otro desarrollo del sistema CP/M que ha introducido Digital Research es el CP/NET. Esta es una versión del CP/M para red que permite que varios usuarios de máquinas diferentes se comuniquen entre sí. A diferencia del MP/M, el CP/NET no requiere un sistema maestro con terminales subordinados, si bien hay un *nudo maestro* que controla la red. Además, este nudo maestro debe operar bajo MP/M y tener conectadas unidades de disco



Bernard Jennings

Contemplando las estrellas
El mapa interplanetario de los asistentes a las fiestas muestra las rutas hiperespaciales entre los planetas y sus respectivas duraciones. Se puede viajar entre planetas que no estén interconectados directamente por rutas hiperespaciales, pero los viajes realizados sin usar las rutas de la red consumen 1 000 unidades de tiempo cada uno. La tarea del programa *GENE* consiste en hallar las rutas más rápidas a través de la red, visitando cada uno de los veinte planetas

Código genético

Continuamos nuestra investigación acerca del aprendizaje de las máquinas

Cuando los científicos dedicados al estudio de la inteligencia artificial observaron la naturaleza en busca de ideas en las cuales basarse para diseñar sistemas que se mejoraran a sí mismos, tres cosas captaron su atención: el sistema nervioso, el sistema inmunológico y el proceso evolutivo. El sistema nervioso, en especial el cerebro humano, es un mecanismo de aprendizaje maravillosamente eficaz. Pero sus operaciones todavía son misteriosas: hemos de comprenderlo lo suficientemente bien para poder copiarlo. Se puede decir que también el sistema de respuesta inmunológica del cuerpo aprende, en tanto y en cuanto llega a distinguirse a sí mismo de otros, de modo que puede atacar y destruir cuerpos extraños. En el transcurso de una vida aprende a reconocer millones de millones de proteínas diferentes y, sin su sorprendente adaptabilidad y fiabilidad, moriríamos rápidamente. Y se trata de un sistema de memoria por repetición mecánica: sus poderes de generalización son rudimentarios.

El sistema evolutivo es, ciertamente, eficaz como medio de crear organismos cada vez más avanzados: puede que sea un poco lento para nuestros fines, pero puede ser acelerado en la simulación por ordenador. Por sobre todo, se lo comprende relativamente bien y es suficientemente simple como para que lo copiemos con ciertas esperanzas de éxito.

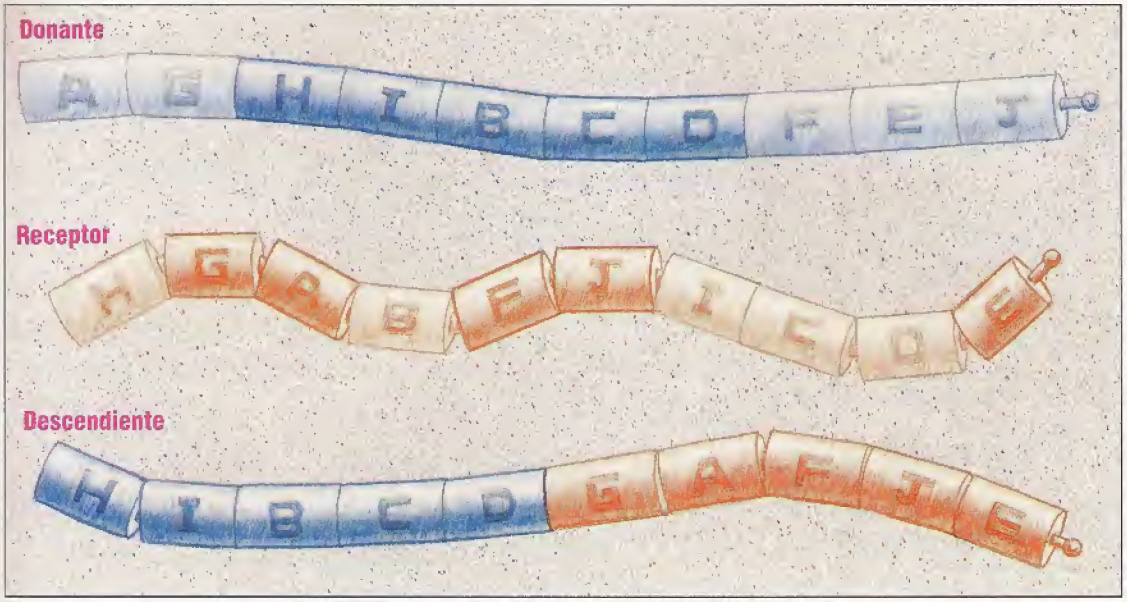
En el capítulo anterior explicamos que un enfoque "darwiniano" al aprendizaje de la máquina tenía algunas ventajas teóricas. Ahora veremos lo que sucede cuando lo ponemos en práctica.

Para ilustrar el aprendizaje de la máquina utilizando un algoritmo evolutivo, analizaremos una versión del "problema del viajante de comercio" (TSP: *travelling salesman problem*). Este problema suele plantearse en el contexto de las 48 capitales de estado del sector continental de Estados Unidos (excluyendo Alaska y Hawai). El vendedor posee una tabla de las distancias entre las ciudades y ha de visitar cada una de las ciudades una sola vez y regresar a su punto de partida. El objetivo es minimizar la distancia total recorrida.

Parece engañosamente simple, ¡pero la cantidad de posibles recorridos o rutas es (N-1)!, donde N es la cantidad de ciudades. En un recorrido de 48 ciudades, las primeras paradas posibles son 47, seguidas de 46 posibles segundas paradas, seguidas de 45 posibles terceras paradas, y así sucesivamente. En realidad, ¡hay más rutas potenciales que átomos en el universo! Aun con 20 ciudades a visitar, ¡la cantidad de recorridos potenciales es de más de ciento veinte millones de millones!

Existen varios enfoques al TSP. Se lo puede tratar como un problema de búsqueda, empleando procedimientos similares a los que describimos en el segundo capítulo de esta serie. También se lo puede manejar con métodos Monte Carlo de acierto o error aleatorio. Sin embargo, vamos a seguir el poco ortodoxo enfoque de considerarlo como un problema de aprendizaje. Queremos un método que explore la enorme cantidad de soluciones potenciales de una forma económica. No esperaremos que nos ofrezca la solución óptima, pero sí esperamos hallar una buena solución.

El programa que ofrecemos aquí, *GENE* (*general evolutionary network explorer*: explorador de redes general y evolutivo) es un sistema de aprendi-



Kevin Jones

Un buen vástago
GENE cría nuevas rutas de la red emparejando dos buenas rutas ya existentes. Dado que éstas están retenidas como series en BASIC de 20 caracteres, el proceso de producción de un descendiente es esencialmente una sencilla manipulación de series. En este ejemplo, se toma de uno de los padres (el donante) la subserie MIBCD y se empalma con el otro padre (el receptor), asegurando de que no se duplique ninguna letra. Este proceso asegura que, al igual que en los verdaderos cruces genéticos, las características de los padres pasen a sus descendientes

zaje adaptado específicamente para explorar redes y desarrollar reglas que evolucionen para producir rutas cada vez más económicas a través de la red. Antes de pasar a describir de forma detallada cómo funciona el programa, necesitamos desarrollar una red a explorar. Podemos tomar un buen ejemplo de *The gatecrasher's guide to the galaxy* (Guía de la galaxia para intrusos), que lista, entre otras cosas, todos los planetas existentes en un radio de 80 años luz en los que usted puede encontrar una fiesta animada el sábado por la noche. Con la ayuda de este mapa podemos transformar el TSP en el PPCD (*planetary party crawl dilemma*: dilema de la ronda por las juergas interplanetarias), en el cual el objetivo es hacer acto de presencia en las 20 fiestas y regresar, esa misma noche, a su punto de partida.

El mapa ilustra las principales rutas hiperespaciales de nuestra localidad galáctica, con los tiempos que lleva el viaje entre cada parada. A los nudos que no están conectados por supercarreteras hiperespeciales se les otorga de forma arbitraria un costo de recorrido de 1 000 unidades de tiempo.

Aprendizaje evolutivo

En un sistema de aprendizaje evolutivo clásico hay una población de estructuras que se tratan como "seudoorganismos". Cada una de estas estructuras (a las que nos referiremos como *reglas*) define una solución potencial al problema en cuestión. También se utilizan para propagar nuevas estructuras (la *descendencia*) en formas que imitan algunas de las características de la reproducción biológica, tales como la transmisión de algunas de las características de los padres a sus hijos.

Dependiendo de su rendimiento en la tarea, se seleccionan reglas que es probable que sobrevivan durante el mayor tiempo y que tengan las mayores probabilidades de reproducirse. En *GENE*, los organismos/reglas tienen una estructura muy simple. Se retienen como series en el programa en BASIC. Por ejemplo, está:

```
ABJHMNCDKTSFRQEGILOP
```

en donde cada letra representa un planeta de la red planetaria y se produce una sola vez en toda la

Ingeniería alfabética

Primero se eligen al azar dos estructuras padre tomadas de entre las que han sobrevivido al proceso de selección (lo que implica, por tanto, que los padres deben ser más aptos para la tarea que la media). Una de estas estructuras se denomina *donante* (R1 %) y la otra "receptor" (R2 %) (ver líneas 3030-3060).

Se toma al azar un trozo de "material genético" (en realidad, una subserie) del donante y se coloca en la serie SS. Luego se llama a la subrutina 3300, que empalma el trozo del donante con el receptor, tomando todos los caracteres de la serie del receptor (excepto aquellos ya presentes en el donante) por el orden en el cual aparecen. El proceso de emparejamiento es, por tanto, asimétrico. Emparejar X con Y no produce el mismo resultado que emparejar Y con X, aunque las posiciones aleatorias P1 % y P2 % sean idénticas.

Veamos someramente un ejemplo a escala reducida. Dados los dos padres:

```
Donante:   A G H I B C D F E J
Receptor:  H G A B F J I C D E
```

podemos seleccionar la subserie

```
H I B C D
```

como la contribución del donante, y empalmar las letras restantes del receptor para producir:

```
H I B C D G A F J E
```

como resultado. Cabe consignar que ésta no es la única forma posible de cruzar un par de reglas-series. Quizá usted pueda pensar en otras. Pero asegura (como en los verdaderos cruces genéticos) que trozos de información de los padres pasen a la siguiente generación. También asegura que cada emparejamiento produzca un descendiente "válido". Esto significa que la rutina de limpieza de la línea 4000 del programa *GENE* no tiene razón de ser, pero la incluimos aquí para ofrecer un esbozo completo de éste.

serie. Por tanto, cada serie de 20 carácteres es una permutación de los planetas a visitar y define una ruta determinada alrededor de la red.

Es simple evaluar cada serie sumando la distancia necesaria para visitar los planetas por el orden especificado. Cuanto menor sea la distancia combinada para completar el recorrido, mejor será la ruta.

Disponemos que las series que sean peores que la media se supriman al cabo de cada "generación". Entonces se plantea el problema de cómo reemplazarlas. Obviamente, si seguimos la analogía biológica, llevaríamos a cabo algo similar a la reproducción sexual para sustituir las series descartadas. Pero la reproducción sexual no es el único medio de generar series nuevas. Alrededor del 8 % de las series supervivientes sufren mutaciones.

La subrutina de mutación, que empieza en la línea 3500, se limita a realizar una cierta cantidad de cambios al azar. Sin embargo, la mutación no es el operador genético primario, sino un operador de fondo que asegura que el sistema no se quede bloqueado en un nivel óptimo local. Las mejoras que se pueden conseguir en una generación sucesiva mediante métodos reproductivos puramente sexuales podrían tener un límite.

Si usted experimenta por sí mismo con el programa, descubrirá que el estándar medio de la población de reglas sí mejora con el tiempo, aunque esta progresión no es continua, porque hasta las mejores reglas a la larga pueden "morir". El programa en realidad hace una trampa, al preservar la mejor regla y sustituirla solamente por una aún mejor.

Usted puede "afinar" el sistema jugando con el coeficiente de muerte representado en la línea 2520. En esta línea, el elemento aleatorio establece el coeficiente de supervivencia de las reglas buenas de una generación a la siguiente en el 88 %, pero éste se puede alterar. Asimismo, puede regular el coeficiente de mutación en la línea 3520, así como introducir operaciones de mutación alternativos, tales como la inversión completa de una regla-serie. Subsisten dos preguntas:

1. ¿Cuán bueno es el método?
2. ¿Por qué funciona?

La primera pregunta se puede responder a través de la comparación con un enfoque Monte Carlo puro. Todos los algoritmos genéticos se modifican de hecho por procedimientos Monte Carlo. En un método Monte Carlo puro, se generan soluciones aleatorias y se conserva la mejor solución, todo ello dentro de un límite de tiempo específico. En este ejemplo, ello supondría probar una regla-serie aleatoria, evaluarla, conservarla si es la mejor disponible hasta ahora, y someterla a una mutación. El proceso repitiría todas las veces que se deseara.

Si usted describe un programa como éste, descubrirá que rápidamente halla una solución bastante buena y que, a medida que transcurre el tiempo, se va mejorando a un nivel menor. Es probable que, al cabo de 20 000 intentos, la mejor solución sea sólo marginalmente mejor, en el mejor de los casos, que la solución tras 10 000 intentos. De forma global, los algoritmos genéticos mejoran para duraciones mayores y si consideramos el por qué de esto nos aproximaremos aún más a la respuesta para la segunda pregunta.

Un método Monte Carlo puro es esencialmente una búsqueda ciega. Un algoritmo genético, por el contrario, se vale de lo que encuentra para continuar la búsqueda. Los patrones específicos que contribuyen a un buen rendimiento se preservan y se propagan a través de la base de conocimiento (la población de reglas) y se vuelven a combinar en contextos ligeramente diferentes. De hecho, la búsqueda se dirige preferentemente hacia regiones del "espacio (multidimensional) del problema" en donde se hayan hallado buenos resultados. A menos que la función de evaluación sea extraordinariamente discontinua, cabe esperar que esto conduzca a una razonable estrategia de búsqueda.

El programa GENE

Las principales estructuras de datos que se utilizan para implementar nuestro sistema de aprendizaje evolutivo corresponden a las siguientes matrices:

NOMBRE$(TAMAÑO%)	Nombres de los nudos de la red
ENLACE%(TAMAÑO%, TAMAÑO%)	Distancia entre los nudos
NX%(TAMAÑO%)	Utilizada en la rutina de emparejamiento
R$(NR%)	Las reglas propiamente dichas (series)
RV(NR%)	Los valores de cada regla

NX% se emplea cuando se empalma la contribución del receptor a la nueva regla que se está creando (rutina 3300), de modo tal que cada nudo/letras aparezca una vez en el descendiente.

R$ retiene la actual población de reglas como series de longitud TAMAÑO%. Cada una define una recorrido particular.

RV da el "valor" de cada regla correspondiente a la distancia del recorrido que representa. Si RV(R%)=0, la regla R% estará "muerta" y será necesario sustituirla en la siguiente generación. (No son posibles recorridos de costo cero.)

TAMAÑO% se puede alterar si usted desea probar mapas de redes diferentes, en cuyo caso necesitará alterar desde la línea 8000 en adelante. NR% se puede ajustar para ver el efecto de una o más reglas sobre la población

```
  10 REM ********************
  11 REM **        GENE      **
  13 REM ********************
 100 GOSUB 1000 : REM preparar matriz mapa
 101 G%=0: SN%=0
 105 B=TAMAÑO%*1000: B$=""
 110 INPUT "Cuantas generaciones?", MG%
 111 GOSUB 1700 : REM reglas iniciales.
 115 IF SN%=0 THEN INPUT "El nudo de partida es el No. ",SN%
 120 REM **** BUCLE PRINCIPAL DEL PROGRAMA ****
 130 G%=G%+1
 140 PRINT "Generación ";G%
 150 GOSUB 2000 : REM evaluar reglas
 160 GOSUB 2500 : REM matar reglas malas
 170 GOSUB 3000 : REM emparejar reglas buenas
 180 GOSUB 3500 : REM mutaciones
 190 GOSUB 4000 : REM limpieza
 200 IF G%<MG% THEN 120
 220 GOSUB 5000 : REM volcar reglas nuevas
 250 END
 999 :
1000 REM — RUTINA PARA PREPARAR MAPA-RED:
1001 TAMAÑO%=20: NR%=24
1010 DIM NOMBRE$(20), ENLACE%(20,20)
1011 DIM NX%(TAMAÑO%)
1012 DIM R$(NR%),RV(NR%)
1013 REM reglas y valores de las reglas
1015 FOR I%=1 TO TAMAÑO%
1020 FOR J%=1 TO TAMAÑO%
1022 ENLACE%(I%,J%)=1000 : REM detec
1023 IF I%=J% THEN ENLACE%(I%,J%)=0
1025 NEXT: NEXT
1030 NC%=0: L%=0
1032 FOR I%=1 TO NR%: RV(I%)=0: NEXT
1033 RESTORE
1040 REM **** LEER NOMBRE Y NUM DE
1050 READ N$, ID%
1055 PRINT N$, ID%
1060 IF ID%< >0 THEN GOSUB 1500
1070 IF ID%<>0 THEN 1040
1080 PRINT NC%;" LECTURA POSICIONES
1088 PRINT L%;" enlaces no por defecto."
1090 RETURN
1100 :
1500 REM — NUDO INDIVIDUAL Y CONEX
1510 NC%=NC%+1
1520 IF ID%<> NC% THEN PRINT"ATENCI
1530 NOMBRE$(NC%)=N$
1550 REM **** PREPARAR MATRIZ DISTA
1560 READ NI%,NT%
1570 ENLACE%(ID%,NI%)=NT%
1588 REM los enlaces cero no importan.
1590 L%=L%+1
1600 IF NI%>0 THEN 1550
1610 RETURN
1620 :
1700 REM — Reglas ficticias iniciales:
1710 S$=LEFT$("ABCDEFGHIJKLMNOPQRS
1715 GOSUB 1780 : REM leer archivo si hay
1720 FOR R%= 1 TO NR%
1730 R$(R%)=S$
1740 PRINT S$;R%
1750 I%=INT(RND(1)*TAMAÑO%+1): J%=
1760 GOSUB 6000 : REM TRUEQUE
1770 NEXT
1775 RETURN
1777 :
1780 INPUT "Archivo de reglas viejo (RETU
1790 IF RF$="" THEN RETURN
1800 F%=OPENUP(RF$)
1810 INPUT # F%, B$, B, SN%
1820 CLOSE # F%
1825 S$=B$
1830 REM solo el de arriba.
1840 RETURN
1850 :
2000 REM — Rutina de evaluacion de reglas
2010 T=0
2020 FOR R%=1 TO NR%
2030 IF RV(R%) <=0 THEN GOSUB 2200
2040 T=T+RV(R%)
2050 NEXT
2060 AV=T/NR% : REM valor medio
2070 PRINT"Marcador medio = ";AV
2080 RETURN
2100 :
2200 REM — Evaluacion de una sola regla
2210 S$=R$(R%)
2220 P1%=SN% : REM nudo de partida
2230 GT%=0
2240 FOR S%=1 TO LEN(S$)
2250 P2%=ASC(MID$(S$,S%,1))-64
2260 IF P2%=SN% THEN GOTO 2290
2270 GT%=GT%+ENLACE%(P1%,P2%)
2280 P1%=P2%
2290 NEXT
2300 RV(R%)=GT%+ENLACE%(P2%,SN%
2310 RETURN
2320 :
2500 REM — Rutina para matar reglas mala
2510 FOR R%=1 TO NR%
2515 IF RV(R%) < B THEN B=RV(R%): B$=
2520 IF RV(R%) > AV OR INT(RND(1)*100-
     RV(R%)=0
2530 NEXT
2540 RETURN
2550 REM — Son mejores los valores men
2560 :
3000 REM — Rutina de emparejamiento
3010 FOR R%=1 TO NR%
3020 IF RV(R%)> 0 THEN GOTO 3120
3030 R1%=INT(RND(1)*NR%+1):IF RV(R
3050 R2%=INT(RND(1)*NR%+1):IF RV(R
3070 REM "padres"elegidos.
3075 P2%=INT(RND(1)*(TAMAÑO%-1)-
3080 P1%=INT(RND(1)*(TAMAÑO%+1)
3090 S$=MID$(R$(R1%),P1%,P2%)
3100 GOSUB 3300 : REM empalmar resto
3110 R$(R%)=S$
3120 NEXT
3140 RETURN
3150 :
3300 REM — Rutina empalme de genes'
3310 FOR S%=1 TO TAMAÑO%
3320 NX%(S%)=0: NEXT
3330 FOR S%=1 TO LEN(S$)
3340 SX%=ASC(MID$(S$,S%,1))-64
3350 NX%(SX%)=NX%(SX%)+1
3360 NEXT
3370 FOR S%=1 TO LEN(R$(R2%))
3380 SX%=ASC(MID$(R$(R2%),S%,1
3390 IF NX%(SX%)>0 THEN GOTO 342
3400 S$=S$+MID$(R$(R2%),S%,1)
```

```
3410 NX%(SX%)=NX%(SX%)+1
3420 NEXT
3440 RETURN
3450 :
3500 REM — Rutina de mutacion:
3510 FOR R%=1 TO NR%
3520 IF RND(100)>8 THEN GOTO 3580
3522 S$=R$(R%)
3525 FOR T%=1 TO 7
3530 R1%=INT(RND(1)*TAMAÑO%+1)
3540 R2%=INT(RND(1)*TAMAÑO%+1)
3560 I%=R1%:J%=R2%:GOSUB 6000:REM TRUEQUE
3565 NEXT T%
3570 R$(R%)=S$
3575 RV(R%)=0
3580 NEXT
3590 REM por ahora solo trocar.
3595 REM tambien necesita inversion.
3600 RETURN
3620 :
4000 REM — Rutina de limpieza:
4010 RETURN
4020 REM ficticia por ahora.
4040 :
5000 REM — Impresion de resultados:
5010 PRINT"Las rutas son:"
5020 BR=TAMAÑO%+1000
5030 R%=0
5040 FOR I%=1 TO NR%
5050 IF RV(I%)=0 THEN 5100
5060 PRINT I%,RV(I%)
5070 PRINT R$(I%)
5080 IF RV(I%)<BR THEN BR=RV(I%): R%=I%
5100 NEXT
5105 A$=GET$
5110 PRINT
5120 PRINT"La mejor es:"
5130 PRINT R$(R%),R%
5131 S$=R$(R%):GOSUB 6400
5133 PRINT"Distancia = ";RV(R%)
5134 A$=GET$
5135 PRINT"La mejor de todas :"
5136 PRINT B$
5140 S$=B$:GOSUB 6400
5143 PRINT"Distancia 8 ";B
5144 A$=GET$
5145 PRINT
5148 GOSUB 5500: REM vuelco archivo
5150 RETURN
5160 :
5500 REM — Rutina de vuelco
5510 INPUT "Nuevo archivo de reglas (RETURN si (no hay) ninguno) " RF$
5520 IF RF$=" " THEN RETURN
5530 F%=OPENOUT(RF$)
5540 PRINT # F%,B$,B,SN%
5550 CLOSE # F%
5555 REM solo la mejor.
5560 RETURN
5570 :
6000 REM **** TROCAR DOS CARACTERES DE S$ ****
6040 IF I%>J% THEN T%=I%: I%=J%: J%=T%
6050 X$=MID$(S$,I%,1)
6060 Y$=MID$(S$,J%,1)
6070 S$=LEFT$(S$,I%-1)+Y$+MID$(S$,I%+1)
6080 S$=LEFT$(S$,J%-1)+X$+MID$(S$,J%+1)
6090 RETURN
6100 :
6400 REM **** VIAJE ****
6430 PRINT " 0 ";NOMBRE$(SN%)
6440 FOR I%=1 TO LEN(S$)
6450 N%=ASC(MID$(S$,I%,1))-64
6460 IF N%<>SN% THEN PRINT I%;" ";NOMBRE$(N%)
6470 NEXT
6480 PRINT I%;" ";NOMBRE$(SN%)
6490 RETURN
6500 :
8000 REM — DATOS PARA MAPA INTERPLANETARIO:
8010 DATA HELIOSOL, 1
8020 DATA 2,4, 17,30, 20,80, 0,0
8030 DATA NIKE,2
8040 DATA 1,4, 3,6, 0,0
8050 DATA APHRODITE,3
8060 DATA 2,6, 5,7, 0,0
8070 DATA LUNA,4
8080 DATA 5,1, 6,10, 0,0
8090 DATA TERRA FIRMA,5
8100 DATA 3,7, 7,8, 6,8, 4,1, 0,0
8110 DATA DEMON KINGDOM,6
8120 DATA 5,8, 7,1, 9,12, 8,12, 5,10, 0,0
8130 DATA PHOBIA, 7
8140 DATA 5,8, 9,13, 8,11, 6,1, 0,0
8150 DATA EUREKA, 8
8160 DATA 6,12, 7,11, 9,1, 10,16, 11,25, 0,0
8170 DATA GALILEO, 9
8180 DATA 10,15, 8,1, 6,12, 7,13, 0,0
8190 DATA TITANIUM CITY, 10
8200 DATA 9,15, 12,24, 11,20, 8,16, 0,0
8210 DATA UMBRIA, 11
8220 DATA 8,25, 10,20, 12,17, 13,28, 0,0
8230 DATA TRIDENT, 12
8240 DATA 10,24, 14,22, 13,20, 11,17, 18,2, 0,0
8250 DATA LIMBO, 13
8260 DATA 12,20, 14,1, 16,48, 15,23, 11,28, 0,0
8270 DATA SUNSET STRIP, 14
8280 DATA 12,22, 13,1, 18,25, 0,0
8290 DATA HADES, 15
8300 DATA 13,223, 16,232, 0,0
8310 DATA TROGSTAR BETA, 16
8320 DATA 13,48, 17,64, 15,32, 0,0
8330 DATA MAXIMA CENTAURI, 17
8340 DATA 16,64, 1,30, 19,88, 0,0
8350 DATA POSEIDON, 18
8360 DATA 12,2, 14,25, 13,25, 0,0
8370 DATA ULTIMA THULE, 19
8380 DATA 17,88, 20,96, 0,0
8390 DATA OMEGA SOLARIS, 20
8400 DATA 1,80, 19,95, 0,0
8410 DATA NINGUNLUGAR, 0
```

Complementos al BASIC

El programa *GENE* está escrito para el BBC Micro.

Commodore 64:

Introduzca las siguientes modificaciones:

```
1820 CLOSE 2
5105 GET A$:IF A$=" " THEN 5105
5134 GET A$:IF A$=" " THEN 5134
5144 GET A$:IF A$=" " THEN 5144
5550 CLOSE 2
```

Para sistemas de disco añada estas líneas:

```
1800 OPEN 2,8,2,RF$+",S,R":F%=2
5530 OPEN 2,8,2,RF$+",S,W":F%=2
```

o éstas para sistemas de cassette:

```
1800 OPEN 2,1,0,RF$:F%=2
5520 OPEN 2,1,1,RF$:F%=2
```

Spectrum:

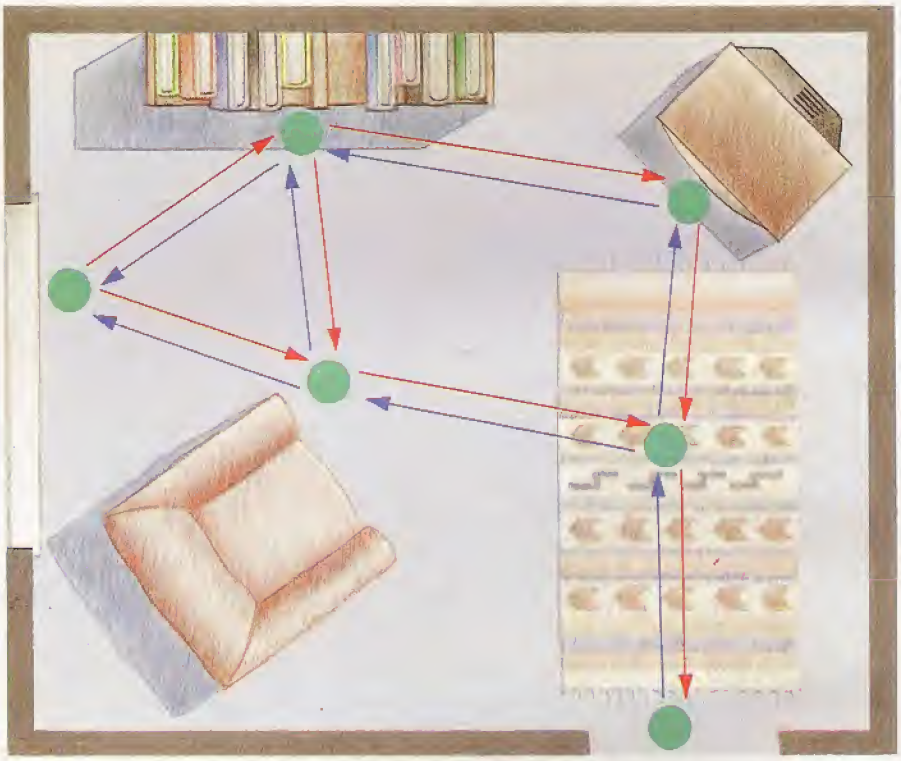
Suprima el signo % en todas las variables, sustituya NOMBRE$() por N$(), ENLACE%(,) por L(,), NX%() por N(), RV() por R(), TAMAÑO% por SI y RF$ por F$ en todo el listado, y encierre entre comillas los nombres de los planetas de las líneas DATA. Introduzca las siguientes modificaciones:

```
1010 DIM N$(20,15),L(20,20)
1012 DIM R$(NR,20),R(NR),D$(3,25)

1710 LET S$="ABCDEFGHIJKLMNOPQRST
     UVWXYZ",(TO SI)

1800 LOAD F$ DATA D$()
1810 LET B$=D$(1)
1820 LET B=VAL(D$(2)):LET SN=VAL(D$,3))

2250 LET P2=CODE S$(S TO S)-64
3090 LET S$=R$(R1)(P1 TO P2-P1)
3340 LET SX=CODE S$(S TO S)-64
3400 LET S$=S$+R$(RS)(S TO S)

5105 IF INKEY$=" " THEN GO TO 5105
5134 IF INKEY$=" " THEN GO TO 5134
5144 IF INKEY$=" " THEN GO TO 5144

5530 LET D$(1)=B$:LET D$(2)=STR$(B)
5540 LET D$(3)=STR$(SN)
5550 SAVE F$ DATA D$()

6050 LET X$=S$(I TO I)
6060 LET Y$=S$(J TO J)
6070 LET S$=S$(TO I-1)+Y$+S$(I+1 TO)
6080 LET S$=S$(TO J-1)+X$+S$(J+1 TO)

6450 N=CODE S$(I TO I)-64
```

Prueba de capacidad

Espacio para moverse
El entorno del robot es un conjunto de posiciones "conocidas", que se han entrado como hechos en su base de datos. Éstas se enlazan mediante un conjunto de "caminos" a lo largo de los cuales se puede desplazar y que se almacenan en el formato "camino(tv,estantería)", y así sucesivamente. En nuestro programa *Simulación de robot móvil* ilustramos cómo utiliza el robot este conocimiento para trazar un camino desde A hasta B

Concluiremos esta serie dedicada al PROLOG con una evaluación de la capacidad del lenguaje para utilizar sus propios programas como datos e ilustrando su idoneidad para la programación de inteligencia artificial



Los japoneses han elegido al PROLOG como "lenguaje central" para su proyecto de ordenadores de quinta generación. Ello se debe a dos motivos fundamentales: Los términos del PROLOG pueden tener una forma muy similar a las "relaciones" de una base de datos relacional, y para los japoneses la base de su sistema se considera una sofisticada máquina de base de datos.

En segundo lugar, el PROLOG es un lenguaje ideal para escribir programas de inteligencia artificial (AI). Ello se debe a su capacidad para usar sus programas como datos, y a que su intérprete se parece a los "motores de inferencia" de muchos sistemas modernos.

Para ver la utilidad que puede tener utilizar sentencias de programas como datos, ofrecemos un programa que es una sencilla simulación de un robot móvil que se desplaza por la habitación de una casa. Este programa ilustra las características más avanzadas del PROLOG y nos dará una noción del aspecto que tienen los programas en este lenguaje y cómo se comportan.

La habitación se define como un conjunto de lugares y un conjunto de caminos que los unen. Queremos que nuestro robot obedezca instrucciones, tales como "ir desde el televisor hasta el sillón", y que encuentre por sí mismo la ruta más corta. En primer lugar, consideremos el objetivo ir(Allí), que significa: "ir desde donde estés hasta Allí por la ruta más corta posible".

En el programa hay dos cláusulas para ir(Allí). La primera es para detectar el caso trivial planteado cuando el robot ya está Allí. Observe que para esto necesitamos un hecho de la base de datos (en(puerta)), que registra la posición actual del robot. La segunda cláusula utiliza este hecho para establecer una posición de partida, y después llama a otro procedimiento: ir(Lugar1,Lugar2).

Dado que ir(Lugar1,Lugar2) se puede usar separado de ir(Lugar), primero comprueba que Aquí y Allí sean lugares que conoce y que Aquí sea en realidad la posición actual. Si falla alguna de estas comprobaciones, el PROLOG abandona la cláusula actual y pasa a la siguiente. La segunda cláusula ir(Aquí,Allí) está para trasladar al robot a Aquí en el caso de que no se hallara ya en esa posición. Esto lo hace con una llamada a ir(Aquí) y, cuando ésta ha triunfado, con otra llamada a ir(Allí). La lectura procesal de esta cláusula es: "para ir desde (Aquí) hasta (Allí) si no estás ya (Aquí), primero ir (Aquí) y luego ir (Allí)". La tercera cláusula para ir(Aquí,Allí) se incluye sólo para imprimir un mensaje de error si no se conoce ninguno de los dos lugares.

El predicado "Findall"

Suponiendo que triunfen las comprobaciones preliminares para el primer ir(Aquí,Allí), el siguiente subobjetivo será findall. Éste es un predicado que en ocasiones ya está incorporado en el PROLOG pero que, de no ser así, se puede añadir (al igual que el FORTH, el PROLOG se puede ampliar fácilmente definiendo nuevos predicados, puesto que los predicados definidos por el usuario poseen el mismo status que los predicados del sistema). Findall posee tres argumentos. El primero es un nombre de variable, el segundo es un objetivo y el tercero es una lista de variables (RL).

Funciona en PROLOG intentando demostrar el objetivo que se le ha dado (plan(Aquí,Allí,[],Ruta), en este caso) todas las veces que pueda. La variable del primer argumento debe concordar con alguna de las del objetivo y, cada vez que éste triunfe, el valor que se había concretado para esa variable se añade a la lista del tercer argumento.

Por ejemplo, el objetivo plan(Lugar1,Lugar2,[],Ruta) hallará una ruta entre Lugar1 y Lugar2 y la colocará en la variable Ruta. De modo que, en este caso, findall reúne todas las rutas que se puedan hallar entre Aquí y Allí y las coloca en una lista.

Llegados a este punto, hemos de destacar que aún no hemos necesitado definir el procedimiento para findall. De hecho, es una práctica normal cuando se programa en PROLOG desarrollar programas de esta forma. Debido a su naturaleza declarativa, escribimos nuestros procedimientos "de arriba hacia abajo", definiendo primero los objetivos de alto nivel y rellenando los detalles.

El predicado shortest(RL,Ruta Corta) es otro predicado que todavía no hemos definido aquí. Su tarea consiste en tomar una lista de listas (Lista1) y crear una

nueva lista (Lista2) que contenga sólo la lista más corta que encuentre. Puesto que todas las listas de Lista1 son rutas, la lista más corta será la ruta más corta.

Hemos llegado a una etapa en la que nuestro robot ha planeado la ruta más corta para llegar a su destino; todo cuanto resta por hacer es que se desplace hasta allí. Esto se consigue mediante otro procedimiento "ficticio", caminar(Ruta). Hemos definido caminar de modo que simplemente escriba las rutas, pero la ruta en realidad es una lista de objetivos del PROLOG. Éstos se le podrían haber dado al PROLOG para que los ejecutara como un programa en lugar de imprimirlos en la pantalla. Se podría definir el predicado mirar(Lugar) de modo que efectuara una exploración por sensor para localizar Lugar y luego hiciera girar al robot, definiendo al mismo tiempo mover(Lugar1,Lugar2) de modo que permitiera que el robot se desplazara entre los dos puntos.

Este programa de desplazamiento lo escribe por sí mismo el programa principal dentro del procedimiento plan. Esto se hace simplemente. En esencia, el algoritmo que utiliza es: para hallar una ruta de A a B, primero hallar una ruta de A a C, después hallar una ruta de C a B. El procedimiento plan es recursivo. La primera cláusula está allí para detener la recursión cuando se alcance el punto de destino (es decir, siempre hay una ruta desde A hasta A).

La segunda cláusula es en realidad la que hace todo el trabajo. Su lectura declarativa es que existe un plan para ir desde A hasta B por la ruta R si:

1. hay un camino desde A hasta C, *y*
2. C no está incluido en la lista ya visitada, V (luego se añadirá C a esta lista para impedir que en el futuro se avance en círculo), *y*
3. hay un plan para ir desde C hasta B por la ruta R1.

Cuando todas estas condiciones son verdaderas, la ruta final, R, es la lista que contiene un programa para ir de A a C añadido a la ruta-hasta-ahora (retenida en R1). Los procedimientos recursivos como éste son muy difíciles de seguir (intente seguirlo mediante lápiz y papel), pero le confieren al PROLOG una extraordinaria concisión.

Para completarlo es preciso actualizar la posición actual del robot. Ello se realiza con los predicados incorporados retract y assert. retract(X) suprime de la base de datos la primera cláusula que concuerda con X, y asserta(X) añade a la base de datos la cláusula X como la primera de ese tipo (assertz(X) añade X como la última). Estos predicados son poderosas herramientas para la programación de AI.

Simulación de robot móvil
Versión en PROLOG estándar:

```
en(puerta).          /*la posición actual del robot*/

ir(Allí):-           en(Allí),write('Ya estoy allí!'),nl,nl,nl.

ir(Allí):-           en(Aquí),ir(Aquí,Allí).

ir(Aquí,Allí):-      lugar(Aquí),lugar(Allí),en(Aquí),findall(Ruta,plan(Aquí,Allí,[],Ruta),RL),shortest(RL,
                     RutaCorta),caminar(RutaCorta),retract(en(Aquí)),asserta(en(Allí)),decirdonde.

ir(Aquí,Allí):-      not(en(Aquí)),write('No estoy en el'),write(Aquí),write('de modo que iré hasta allí
                     primero.'),nl,nl,ir(Aquí),ir(Allí).

ir(Aquí,Allí)        write('Sólo puedo visitar los lugares que sé que existen'),nl,nl.

plan(A,A,_,R).
plan(A,B,V,R):-      camino(A,C),not(member(C,V)),append[C],V,V1),plan(C,B,V1,R1),append([mirar(C),
                     mover(A,C)],R1,R).

caminar(Ruta):-      write(Ruta),nl,nl.      /*a definir por completo luego*/

decirdónde:-         en(Lugar),write('Estoy en el'),write(Lugar),nl,nl,nl.

/*una lista de los caminos que conoce el robot*/     /*y una lista de los lugares que conoce*/

camino(estantería,sillón).                          lugar(puerta).
camino(sillón,estantería).                          lugar(alfombra).
camino(estantería,tv).                              lugar(tv).
camino(tv,estantería).                              lugar(estantería).
camino(tv,alfombra).                                lugar(ventana).
camino(alfombra,tv).                                lugar(sillón).
camino(alfombra,sillón).
camino(sillón,alfombra).
camino(puerta,alfombra).
camino(alfombra,puerta).
camino(ventana,sillón).
camino(sillón,ventana).
camino(ventana,estantería).
camino(estantería,ventana).
```

Versión en Micro-PROLOG:

```
(en puerta)          /*la posición actual del robot*/

((ir X)              (en X) (P Ya estoy allí!) PP,PP,PP)

((ir X)              (en Y) (ir Y X))

((ir X Y)            (lugar X) (lugar Y)(en X)
                     (findall Z (plan XY()Z)x)
                     (shortest x y)
                     (caminar y)
                     (DELCL ((enX))) (ADDCL ((en Y)))
                     (decirdónde))
((ir X Y)            (NOT en X)
                     (P No estoy en el) (PX)
                     (P de modo iré allí primero.) PP PP
                     (ir X) (ir Y))
((ir X Y)            (P Sólo puedo visitar lugar que sé que existen) PP PP)
(plan X X x1 Z)
((plan X Y x Z)      (camino X X1) (NOT member X1 x) (append (X1) xx1)
                     (plan X1 Y x1 Z1)
                     (append ((mirar X1)(mover X X1))Z1 Z))

((caminar Z)         (P Z) PP PP)
((decirdónde)        (en X) (P Estoy en el X) PP PP PP)

/*una lista de los caminos que conoce el robot*/     /*y una lista de los lugares que conoce*/

(camino estantería sillón)                          (lugar puerta)
(camino sillón estantería)                          (lugar alfombra)
(camino estantería tv)                              (lugar tv)
(camino tv estantería)                              (lugar estantería)
(camino tv alfombra)                                (lugar ventana)
(camino alfombra tv)                                (lugar sillón)
(camino alfombra sillón)
(camino sillón alfombra)
(camino puerta alfombra)
(camino alfombra puerta)
(camino ventana sillón)
(camino sillón ventana)
(camino ventana estantería)
(camino estantería ventana)
```

Maniobras programadas
La información acerca del entorno del robot la entra el usuario en la base de datos del PROLOG. El PROLOG utiliza el procedimiento ir(lugar1,lugar2) para construir una ruta desde A hasta B, lo que hace hallando primero una ruta desde A a C y luego otra desde C hasta B. Observe el empleo del símbolo de subrayado como un argumento dentro de la cláusula plan(A,A,_,B). El símbolo se puede emplear en PROLOG como una *variable anónima* o *máscara* para la cual no se concretará ningún valor cuando se ejecute la cláusula. En este caso en particular, la variable anónima se utiliza como parte de una cláusula que existe para mostrar que siempre hay una ruta desde A hasta A

El Nuevo Mundo (III)

Concluimos nuestro proyecto ofreciendo la última parte del listado

El programa, aunque escrito para el Commodore 64, se puede ejecutar en ordenadores Amstrad con la introducción de unas pocas modificaciones menores. Todos los PRINT CHR$(147) se deben sustituir por CLS. Todas las líneas que aguarden una pulsación de tecla, como:

< n.º línea > GET I$:IF I$=" " THEN < n.º línea >

se deben reemplazar por:

< n.º línea > I$=" ": WHILE I$=" ":I$=INKEY$: WEND

Por último, para que se seleccione la modalidad de visualización en pantalla a 40 columnas se debe insertar la siguiente línea:

5 MODE 1

```
6530 REM BOTE SALVAVIDAS
6535 IF M(1)=1 THEN RETURN
6536 PRINTCHR$(147)
6540 M(1)=1
6550 S$="SE HA AVISTADO UN BOTE SALVAVIDAS*":GOSUB 9100
6552 S$="NAVEGANDO A LA DERIVA A LO LEJOS* ":GOSUB 9100
6554 PRINT:GOSUB 9200
6556 S$="A TRAVES DEL CATALEJO VES*":GOSUB 9100
6558 S$="QUE CONTIENE:*":GOSUB 9100
6560 PRINT:GOSUB 9200
6562 S$="4 PERSONAS*":GOSUB 9100
6563 GOSUB 9200
6564 S$="Y UN GRAN COFRE!*":GOSUB 9100
6566 PRINT:GOSUB 9200
6568 S$="SI ALTERAS TU RECORRIDO*":GOSUB 9100
6570 S$="PARA RECOGERLOS*":GOSUB 9100
6572 S$="TARDARAS DOS DIAS MAS*":GOSUB 9100
6574 PRINT:GOSUB 9200
6576 S$="QUIERES RESCATARLOS (S/N)*":GOSUB 9100
6578 INPUT I$:I$=LEFT$(I$,1)
6580 IF I$<>"S" AND I$<>"N" THEN 6578
6585 IF I$="S" THEN 6600
6588 PRINT:GOSUB 9200
6590 S$="EL BOTE SALVAVIDAS DESAPARECE....*":GOSUB 9100
6592 PRINT:GOSUB 9200
6594 S$=K$:GOSUB 9100
6596 GET I$:IF I$=" " THEN 6596
6599 RETURN
6600 PRINT:GOSUB 9200
6610 EW=EW+2/7
6625 IF CN <>16 THEN 6630
6627 S$="NO PUEDES RECOGERLOS*":GOSUB 9100
6628 S$="PORQUE NO TIENES LUGAR EN EL BARCO*":GOSUB
     9100
6629 GOTO 6592
6630 X=16-CN:IF X>3 THEN 6635
6632 S$="EN EL BARCO SOLO TIENES LUGAR PARA*":GOSUB
     9100
6633 PRINT X;"PERSONAS MAS"
6634 PRINT:GOSUB 9200
6635 S$="RECOGES:*":GOSUB 9100
6638 X=0
6640 FOR T=1 TO 16
6645 IF TS(T,1)<>0 THEN 6679
6650 X=X+1
6655 IF X>4 THEN T=16:GOTO 6679
6660 CN=CN+1
6665 TS(T,1)=INT(RND(1)*5)+1
6668 TS(T,2)=INT(RND(1)*50)+50
6670 PRINT"1 ":C$(TS(T,1))
6679 NEXT
6680 PRINT:GOSUB 9200
6682 S$="EL COFRE CONTIENE:*":GOSUB 9100
6685 FOR T=1 TO 4
6690 X=INT(RND(1)*10)+10
6692 PRINT X;U$(T);"S DE ";P$(T)
6693 IF PA(T)=-999 THEN PA(T)=0
6694 PA(T)=PA(T)+X
6695 NEXT
6699 GOTO 6592
```

Quizás una epidemia haga presa de la tripulación...

```
6700 REM AZOTE DE LA EPIDEMIA
6705 IF M(2)=1 THEN RETURN
6706 PRINTCHR$(147)
6710 M(2)=1
6712 S$="SE DECLARA UNA EPIDEMIA*":GOSUB 9100
6714 PRINT:GOSUB 9200
6716 X=1
6718 FOR T=1 TO 16
6720 IF TS(T,1)=2 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN X
     =0:T=16
6722 NEXT
6724 Y=1
6726 IF OA(1)<>0 AND OA(1)<>-999 THEN Y=0
6730 X=((X+Y)*10)+5
```

...y que los tripulantes puedan o no sobreponerse a la enfermedad dependerá de que usted haya contratado a un médico y comprado medicinas antes de zarpar

```
6732 I$="TIENES ":IF X=0 AND Y=0 THEN 6740
6734 IF X=1 THEN S$="SIN NINGUN MEDICO*":GOSUB 9100:I$="NI "
6736 IF Y=1 THEN S$=I$+"NINGUNA MEDICINA*":GOSUB 9100
6740 S$="MUCHOS DE LOS TRIPULANTES ESTAN AFECTADOS*":GOSUB 9100
6745 PRINT:GOSUB 9200
6750 X=0
6755 FOR T=1 TO 16
6756 IF RND(1) <.3 THEN 6775
6760 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 6775
6765 TS(T,2)=TS(T,2)-Z
6770 IF TS(T,2)<1 THEN TS(T,2)=-999:X=X+1
6775 NEXT
6776 IF Y=1 THEN 6780
6777 S$="SE HAN UTILIZADO LA MITAD DE TUS MEDICINAS*":GOSUB 9100
6778 OA(1)=INT((OA(1)/2)+.5)
6780 PRINT:GOSUB 9200
6785 IF X=0 THEN 6797
6790 PRINT"Y";X;
6792 S$="TRIPULANTES MURIERON*"
6794 IF X=1 THEN S$="TRIPULANTE MURIO*"
6795 GOSUB 9100
6796 PRINT:GOSUB 9200
6797 S$=K$:GOSUB 9100
6798 GET I$:IF I$=" " THEN 6798
6799 RETURN
```

El barco puede ser atacado por piratas...

```
6800 REM PIRATAS
6805 IF M(3)=1 THEN RETURN
6810 X=0
6812 FOR T=1 TO 16
6814 IF TS(T,2)=0 OR TS(T,2)=-999 THEN X=X+1
6815 NEXT
6816 IF X=16 THEN RETURN
6818 M(3)=1
6820 PRINT CHR$(147)
```

...¿habrá a bordo algunas armas para defenderse de ellos y reducir las pérdidas al mínimo?

```
6822 S$=" EL BARCO ES ATACADO POR PIRATAS!*":GOSUB 9100
6824 PRINT:GOSUB 9200
6825 K=2
6826 IF OA(2)=0 OR OA(2)=-999 THEN K=4
6828 S$=" A PESAR DE TUS ARMAS*"
6830 IF K=4 THEN S$="NO TIENES ARMAS*"
6832 GOSUB 9100
6835 X=0
6836 FOR T=1 TO 16
6838 IF TS(T,2)=0 OR TS(T,2)=-999 THEN 6845
```

```
6840 X=X+1:TS(T,2)=-999
6842 IF X=K THEN T=16
6845 NEXT
6850 PRINT X;
6855 S$="TRIPULANTE HA RESULTADO MUERTO*"
6856 IF X>1 THEN S$="TRIPULANTES HAN RESULTADO MUERTOS*"
6860 GOSUB 9100
6865 PRINT:GOSUB 9200
6890 S$=K$:GOSUB 9100
6895 GET I$:IF I$=" " THEN 6895
6899 RETURN
6900 REM TIMON
6905 IF M(4)=1 THEN RETURN
6910 PRINT CHR$(147)
6915 M(4)=1
6920 S$="HAY PROBLEMAS CON EL TIMON!*":GOSUB 9100
6925 PRINT:GOSUB 9200
6928 X=4
6930 FOR T=1 TO 16
6935 IFTS(T,1)=3 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN X=1:T=16
6938 NEXT
6940 S$="A PESAR DE QUE TIENES UN MECANICO*"
6945 IF X=4 THEN S$="NO TIENES MECANICO Y*"
6950 GOSUB 9100
6955 S$=TU VIAJE DURARA*":GOSUB 9100
6960 PRINT X;"SEMANAS MAS"
6965 EW=EW+X
6967 PRINT:GOSUB 9200
6969 S$=K$:GOSUB 9100
6970 GET I$:IF I$=" " THEN 6970
6975 RETURN
7000 REM TORMENTA
7005 IF M(5)=1 THEN RETURN
7010 PRINTCHR$(147)
7015 M(5)=1
7020 S$="EL VIENTO TE APARTA DE TU RUMBO*":GOSUB 9100
7022 S$="DURANTE UNA TORMENTA!*:GOSUB 9100
7025 PRINT:GOSUB 9200
7028 X=2
7030 FOR T=1 TO 16
7035 IF TS(T,1)=4 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN X=1:T=16
7038 NEXT
7040 S$=A PESAR DE QUE CUENTAS CON UN OFICIAL*"
7045 IF X=2 THEN S$="NO TIENES NINGUN OFICIAL Y*"
7049 GOTO 6950
```

Si se te están acabando las provisiones, quizá quieras poner rumbo hacia esta isla y reabastecerte, pero al hacerlo tal vez el viaje se prolongue

```
7050 REM ISLA
7055 IF M(6)=1 THEN RETURN
7060 PRINTCHR$(147)
7065 M(6)=1
7070 S$="TUS CARTAS INDICAN LA EXISTENCIA DE UNA ISLA*":GOSUB
     9100
7071 S$="EN LA QUE PODRIAS*":GOSUB 9100
7072 S$="REABASTECERTE DE PROVISIONES*":GOSUB 9100
7073 S$="PERO SI TE DESVIAS HACIA ELLA*":GOSUB 9100
7074 S$="SUPONDRA UNA MAYOR DURACION DEL VIAJE*":GOSUB
     9100
7075 PRINT:GOSUB 9200
7080 S$="QUIERES IR A LA ISLA*":GOSUB 9100
7082 INPUT I$:I$=LEFT$(I$,1)
7084 IF I$<>"S" AND I$<>"N" THEN 7082
7086 PRINT:GOSUB 9200
7090 IF I$="N" THEN 7145
7100 S$="LLEGAS A LA ISLA*":GOSUB 9100
7105 S$="Y CONSIGUES:*":GOSUB 9100
7106 IF B$="N" THEN 7110
7107 PRINT:GOSUB 9200
7108 PRINT"NADA!":GOSUB 9200
7109 S$="(RECUERDA EL ALBATROS!)":GOSUB 9100;GOTO 7130
7110 FOR T=1 TO 4
7112 IF RND(1)<.25 THEN 7129
7115 X=INT(RND(1)*10)+5
7120 PRINT X;U$(T);"S DE";P$(T)
7122 IF PA(T)=-999 THEN PA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="PERO AHORA EL VIAJE DURARA*":GOSUB 9100
7135 X=INT(RND(1)*2)+1
7139 PRINT X;:S$="SEMANAS MAS*":GOSUB 9100
7140 EW=EW+X
7145 PRINT:GOSUB 9200
7150 S$=K$:GOSUB 9100
7155 GET I$:IF I$=" " THEN 7155
7159 RETURN
```

Siete factores se pueden combinar para favorecer la rebelión de la tripulación. Esta rutina genera un factor de amotinamiento, MF, comprobando las condiciones que prevalecen al final de cada semana del viaje y sumándole a MF las estimaciones apropiadas

```
7200 REM MOTIN
7210 MF=0
7215 IF M$="S" THEN MF=MF+30
7220 NC=0
7225 FOR T=1 TO 16
7228 IF TS(T,1)=5 AND TS(T,2)<>0 AND TS(T,2)<>-999 THEN NC=1:T=16
7230 NEXT
7235 IF NC=0 THEN MF=MF+30
7240 IF A$="S" THEN MF=MF-20
7245 IF B$="S" THEN MF=MF+30
7250 IF CN>12 THEN MF=MF+30
7255 IF WT>MO THEN MF=MF+30
7260 IF WK>8 THEN MF=MF+((WK-8)*10)
7275 MF=MF+INT(RND(1)*30)
```

Si el factor de amotinamiento es mayor que 75 (pero menor que 100), se le advierte al jugador del descontento de la tripulación

```
7280 IF MF<75 THEN RETURN
7282 PRINT CHR$(147)
7284 IF MF>100 THEN 7300
7285 S$="LA SITUACION EN EL BARCO*":GOSUB 9100
7286 S$="ESTA EMPEORANDO*":GOSUB 9100
7287 S$="Y ALGUNOS DE LOS TRIPULANTES*":GOSUB 9100
7288 S$="HABLAN YA DE MOTIN!*":GOSUB 9100
7290 PRINT:GOSUB 9200
7292 S$=K$:GOSUB 9100
7294 GET I$:IF I$=" " THEN 7294
7299 RETURN
```

Si el factor de amotinamiento se eleva por encima de 100, la tripulación se subleva, listándose sus quejas antes de dejar al capitán en un bote a la deriva

```
7300 PRINT CHR$(147)
7305 PRINT:GOSUB 9200
7310 S$="LA TRIPULACION SE HA AMOTINADO*":GOSUB 9100
7312 S$="PORQUE:*":GOSUB 9100
7313 X=0
7314 IF M$<>"S" THEN 7320
7315 GOSUB 9200:X=X+1:PRINT X;
7316 S$="HAN ESTADO A MEDIA RACION*":GOSUB 9100
7318 S$=" DURANTE PARTE DEL VIAJE*":GOSUB 9100
7320 IF NC<>0 THEN 7325
7321 GOSUB 9200:X=X+1:PRINT X;
7322 S$="NO HAY COCINERO*":GOSUB 9100
7324 S$=" Y LA COMIDA ES ASQUEROSA*":GOSUB 9100
7325 IF B$<>"S" THEN 7330
7326 GOSUB 9200:X=X+1:PRINT X;
7327 S$="EL ALBATROS FUE ABATIDO!*":GOSUB 9100
7330 IF CN<13 THEN 7335
7331 GOSUB 9200:X=X+1:PRINT X;
7332 S$="LA TRIPULACION ESTA HACINADA*":GOSUB 9100
7335 IF MO>=WT THEN 7340
7336 GOSUB 9200:X=X+1:PRINT X;
7337 S$="NO HAY SUFICIENTE ORO*":GOSUB 9100
7338 S$=" PARA PAGARLES SUS SALARIOS*":GOSUB 9100
7340 IF WK<=8 THEN 7350
7341 GOSUB 9200:X=X+1:PRINT X;
7342 S$="LLEVAN NAVEGANDO*":GOSUB 9100
7343 S$=" MAS DE 8 SEMANAS*":GOSUB 9100
7350 PRINT:GOSUB 9200
7360 S$="LA TRIPULACION SE APODERA DEL BARCO*":GOSUB 9100
7362 GOSUB 9200
7363 S$="Y EMPRENDE EL REGRESO*":GOSUB 9100
7370 GOSUB 9200
7372 S$="DEJANDOTE A TI ABANDONADO*":GOSUB 9100
7373 S$="EN UN BOTE A LA DERIVA*":GOSUB 9100
7374 S$="OJALA QUE ALGUIEN TE RECOJA*":GOSUB 9100
7375 PRINT:GOSUB 9200
7380 PRINT"     FIN DEL JUEGO"
7382 END
```

Estas cortas rutinas se utilizan mucho en el programa para producir efectos tales como retardar la salida por pantalla

```
9100 REM IMPRESION LENTA DE S$
9110 FOR S3=1 TO 32
9115 IF MID$(S$,S3,1)="*" THEN S3=32:GOTO 9140
9120 PRINT MID$(S$,S3,1);
9130 FOR S4=1 TO 25:NEXT
9140 NEXT:PRINT
9199 RETURN
9200 REM BUCLE DE DEMORA
9210 FOR S5=1 TO 1000:NEXT
9299 RETURN
9300 REM REDUCIR FORTALEZA TRIPULACION EN FUNCION DE WF
9310 FOR S1=1 TO 16
9315 IF TS(S1,2)=0 THEN 9340
9320 TS(S1,2)=TS(S1,2)-WF
9330 IF TS(S1,2)<1 THEN TS(S1,2)=-999
9340 NEXT
9399 RETURN
```

El barco llega a su destino...

```
10000 REM LLEGADA AL NUEVO MUNDO
10001 PRINTCHR$(147):GOSUB 9200
10005 S$="LLEGAS AL NUEVO MUNDO*":GOSUB 9100
10006 PRINT:GOSUB 9200
10007 S$="MIENTRAS TE APROXIMAS A LA COSTA*":GOSUB 9100
10009 S$="SALEN LOS NATIVOS EN CANOAS PARA RECIBIRTE*":GOSUB 9100
10010 PRINT:GOSUB 9200
10015 IF OA(2)=0 THEN 10050
10017 S$="SU ASPECTO ES FIERO Y ESTAN ARMADOS!!*":GOSUB 9100
10018 PRINT:GOSUB 9200
10020 S$="ABRES FUEGO? (S/N)*":GOSUB 9100
10022 INPUT I$:I$=LEFT$(I$,1)
10024 IF I$<>"N" AND I$<>"S" THEN 10022
10026 IF I$="N" THEN 10050
10028 PRINT:GOSUB 9200
10030 S$="HAN MUERTO MUCHOS NATIVOS*":GOSUB 9100
10032 S$="PERO DURANTE LA NOCHE*":GOSUB 9100
10034 S$="OTROS REGRESAN*":GOSUB 9100
10036 S$="Y LE PRENDEN FUEGO A TU BARCO!!!!*":GOSUB 9100
10038 PRINT:PRINT:GOSUB 9200
10040 S$="      JUEGO TERMINADO*":GOSUB 9100
10042 END
10044 GOTO 10042
10050 S$="TE LLEVAN A CONOCER A SU*":GOSUB 9100
10052 S$="JEFE. YA HA CONOCIDO ANTES A GENTE DE*":GOSUB 9100
10054 S$="TU RAZA Y SE MUESTRA MUY AMABLE*":GOSUB 9100
10056 GOSTUB 9200
10058 S$="LA TRIPULACION HA COMIDO Y ESTA DESCANSANDO*"-
      GOSUB 9100
10060 PRINT:GOSUB 9200
10062 S$="MAÑANA COMENZARA LA ACTIVIDAD COMERCIAL*":GOSUB 9100
10064 PRINT:GOSUB 9200
10066 S$=K$:GOSUB 9100
10068 GET I$:IF I$="" THEN 10068
10069 RETURN
```

...y comienza la actividad comercial

```
10070 PRINTCHR$(147):GOSUB 9200:REM INTERCAMBIO
10072 IF OA(2)=0 THEN 10080
10074 S$="EL JEFE NO QUIERE TUS ESCOPETAS*":GOSUB 9100
10076 S$="PORQUE PODRIAN ACARREARLE PROBLEMAS*":GOSUB 9100
10078 PRINT:GOSUB 9200
10080 IFOA(3)<>0 OR OA(4)<> 0 OR OA(5)<>0 OR OA(6)<>0 THEN 101000
10085 S$="NO TE QUEDA NINGUNA MERCANCIA*":GOSUB 9100
10090 S$="CON LA QUE COMERCIAR*":GOSUB 9100
10095 GOTO 10038
10100 S$="EN TRUEQUE POR LOS CUCHILLOS*":GOSUB 9100
10102 S$="SAL TELA O JOYAS QUE POSEAS*":GOSUB 9100
10104 S$="TE OFRECE PERLAS, FIGURILLAS*":GOSUB 9100
10106 S$="Y ESPECIAS*":GOSUB 9100
10108 PRINT:GOSUB 9200
10110 S$="CUANDO SALISTE DE PUERTO ESTAS*":GOSUB 9100
10112 S$="VALIAN*":GOSUB 9100
10114 S$="PERLAS - 2 P DE ORO CADA UNA*":GOSUB 9100
10116 S$="FIGURILLAS - 2 P DE ORO CADA UNA*":GOSUB 9100
10118 S$="ESPECIAS - 1 PIEZA DE ORO EL GRAMO*":GOSUB 9100
10120 PRINT:GOSUB 9200
10122 S$="PERO QUIZAS CUANDO VUELVAS A CASA*":GOSUB 9100
10124 S$="QUIZAS ESTOS VALORES HAYAN CAMBIADO*":GOSUB 9100
10125 PRINT:GOSUB 9200:S$=K$:GOSUB 9100
10126 GET I$:IF I$="" THEN 10126
10130 FOR T=3 TO=6
10135 IF OA(T)=0 THEN 10200
10140 PRINTCHR$(147):GOSUB 9200
10145 PRINT"TIENES":OA(T);
10150 IF T=3 THEN S$="SACOS DE SAL*"
10151 IF T=4 THEN S$="BALAS DE TELA*"
10152 IF T=5 THEN S$="CUCHILLOS*"
10153 IF T=6 THEN S$="JOYAS*"
10155 GOSUB 9100
10156 PRINT:GOSUB 9200
10160 S$="A CAMBIO EL JEFE TE OFRECE*":GOSUB 9100
10165 PRINT"YA SEA";OA(T)*EQ(T-2,1);"PERLAS"
10166 PRINT"O BIEN";OA(T)*EQ(T-2,2);"FIGURILLAS"
10167 PRINT"      O";OA(T)*EQ(T-2,3);"GRAMOS DE ESPECIAS"
10168 PRINT:GOSUB 9200
10170 S$="QUIERES PERLAS, FIGURILLAS*":GOSUB 9100
10172 S$="O ESPECIAS?*":GOSUB 9100
10174 S$="(ENTRA 1, 2 o 3)*":GOSUB 9100
10175 INPUT I$
10176 I=VAL(I$):IF I<1 OR I>3 THEN 10174
10180 AO(I(=AO(I)+(OA(T)*EQ(T-2,I))
10190 PRINT:PRINT"LAS ";T$(I);" SE CARGAN EN EL BARCO"
10192 S$=K$: GOSUB 9100
10194 GET I$:IF I$="" THEN 10194
10200 NEXT
10210 PRINT:PRINT:GOSUB 9200
10215 S$="   FIN DEL INTERCAMBIO*":GOSUB 9100
10216 PRINT:GOSUB 9200
10218 S$="HAS OBTENIDO:*":GOSUB 9100
10220 PRINT AO(1);"PERLAS"
10222 PRINT AO(2);"FIGURILLAS"
10224 PRINT AO(3);"GRAMOS DE ESPECIAS"
10226 PRINT:GOSUB 9200
10228 S$=K$:GOSUB 9100
10229 GET I$:IF I$="" THEN 10229
10230 RETURN
```

¿Quieres participar en una revuelta local? Las recompensas son elevadas... pero también lo son los riesgos si el golpe fracasa

```
10300 REM REVOLUCION
10305 IF OA(2)=0 THEN RETURN
10310 PRINTCHR$(147):GOSUB 9200
10315 S$="DURANTE LA NOCHE UN RIVAL DEL*":GOSUB 9100
10316 S$="JEFE VISITA EL BARCO EN SECRETO*":GOSUB 9100
10317 PRINT:GOSUB 9200
10318 S$="QUIERE COMPRAR TUS ESCOPETAS*":GOSUB 9100
10320 S$="PARA UNA REVOLUCION*":GOSUB 9100
10322 PRINT:GOSUB 9200
10324 S$="TE OFRECE 30 PERLAS POR CADA ESCOPETA*":GOSUB 9100
10326 S$="LE VENDES LAS ESCOPETAS? (S/N)*":GOSUB 9100
10328 INPUT I$:I$=LEFT$(I$,1)
10330 IF I$<>"N" AND I$<>"S" THEN 10328
10332 IF I$="S" THEN 10400
10334 PRINT:GOSUB 9200
10336 S$="EL JEFE SE ENTERA Y SE SIENTE*":GOSUB 9100
10338 S$="AGRADECIDO HACIA TI*":GOSUB 9100
10340 S$="TE DA PROVISIONES GRATIS*":GOSUB 9100
10342 S$="PARA EL VIAJE DE REGRESO*":GOSUB 9100
10344 GOSUB 9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="Y 50 PERLAS!!*":GOSUB 9100
10348 AO(1)=AO(1)+50
10350 PRINT:GOSUB 9200
10352 S$=K$:GOSUB 9100
10354 GET I$:IF I$="" THEN 10354
10359 RETURN
10400 PRINTCHR$(147):GOSUB 9200
10405 IF RND(1)<.75 THEN 10450
10410 S$="LA REVOLUCION HA TRIUNFADO*":GOSUB 9100
10412 PRINT:GOSUB 9200
10415 S$="EL NUEVO JEFE TE RECOMPENSA CON*":GOSUB 9100
10420 S$="PROVISIONES GRATUITAS PARA EL VIAJE*":GOSUB 9100
10425 S$="DE REGRESO.*":GOSUB 9100
10429 AO(1)=AO(1)+(OA(2)*30):REM SUMAR PERLAS
10430 OA(2)=0
10431 GOTO 10350
10450 S$="LA REVOLUCION FRACASA!!*":GOSUB 9100
10452 PRINT:GOSUB 9200
10455 S$="EL VIEJO JEFE ESTA ENFADADO CONTIGO*":GOSUB 9100
10457 LE PRENDE FUEGO A TU BARCO Y ROBA*":GOSUB 9100
10458 S$="TODO!!*":GOSUB 9100
10459 PRINT:GOSUB 9200
10460 S$="      JUEGO TERMINADO!!*":GOSUB 9100
10462 END
10464 GOTO 10462
10500 REM FIN DEL VIAJE
```

El barco regresa y se realiza una evaluación de la actuación del jugador como comerciante

```
10501 PRINTCHR$(147):GOSUB 9200
10505 S$="CON UNA TRIPULACION FUERTE Y VIENTOS*":GOSUB 9100
10507 S$="FAVORABLES EL VIAJE DE REGRESO VA*":GOSUB 9100
10508 S$="BIEN Y DURA SOLO 8 SEMANAS*":GOSUB 9100
10512 WW=0
10514 FOR T=1 TO 5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT:GOSUB 9200
10520 S$="FACTURA SALARIAL PARA EL VIAJE DE REGRESO*":GOSUB
      9100
10522 PRINT WW;"PIEZAS DE ORO"
10524 PRINT:GOSUB 9200
10526 S$="CUANDO REGRESAS*":GOSUB 9100
10528 S$="PARA VENDER TUS MERCANCIAS*":GOSUB 9100
10530 S$="ESTAS VALEN ENTONCES*":GOSUB 9100
10532 PRINT"PERLAS - ";V2(1);"PIEZAS DE ORO"
10534 PRINT"FIGURILLAS - ";V2(2);"PIEZAS DE ORO"
10536 PRINT"ESPECIAS - ";V2(3);"PIEZAS DE ORO"
10538 PRINT:S$="OBTIENES UN TOTAL DE*":GOSUB 9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*V2(3))
10542 PRINT X;"PIEZAS DE ORO"
10545 PRINT:S$=K$:GOSUB 9100:PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="AHORA POSEES*":GOSUB 9100
10552 PRINT MO+X;"PIEZAS DE ORO"
10555 PRINT:GOSUB 9200
10556 S$="LA FACTURA SALARIAL PARA EL VIAJE ES DE*":GOSUB 9100
10557 PRINT WT+WW;"PIEZAS DE ORO"
10559 PRINT:GOSUB 9200
10560 S$="TERMINAS EL VIAJE CON:*":GOSUB 9100
10562 Z=MO+X-WT-WW
10565 PRINT Z;"PIEZAS DE ORO"
10566 PRINT:GOSUB 9200
10567 PRINT:S$="TU CLASIFICACION ES:*":GOSUB 9100:PRINT
10568 IF Z>3200 THEN S$="CAPITALISTA DE PRIMER ORDEN*":GOSUB
      9100:END
10569 IF Z>2500 THEN S$="INSIGNE COMERCIANTE*":GOSUB 9100:END
10570 IF Z>2000 THEN S$="MERCACHIFLE DE III CLASE*":GOSUB
      9100:END
10571 IF Z>1000 THEN S$="MAS BIEN UN PRIMO*":GOSUB 9100:END
10572 S$="MAS PATO QUE PIRATA*"
10573 GOSUB 9100:END
```

Metro de sastre

Analicemos las rutinas ROM del OS del Spectrum que se encargan del sistema de archivo en cinta

Un Spectrum de Sinclair normal está provisto de sólo una interface para cinta que permite guardar y cargar programas y datos. Pero la adición de la Interface 1 da acceso a los microdrives, la interface serial y las redes de área local. Esto se considera como sistemas alternativos de archivo un poco como el BBC Micro posee sus propias alternativas. Sin embargo, en el BBC Micro se selecciona el sistema de archivo por medio de una instrucción *, mientras que en el Spectrum empleamos una sintaxis diferente de instrucciones para diferenciar las referidas al sistema de archivo en cinta de otras pertenecientes a otros sistemas de archivo disponibles para este ordenador. Por ejemplo, la instrucción

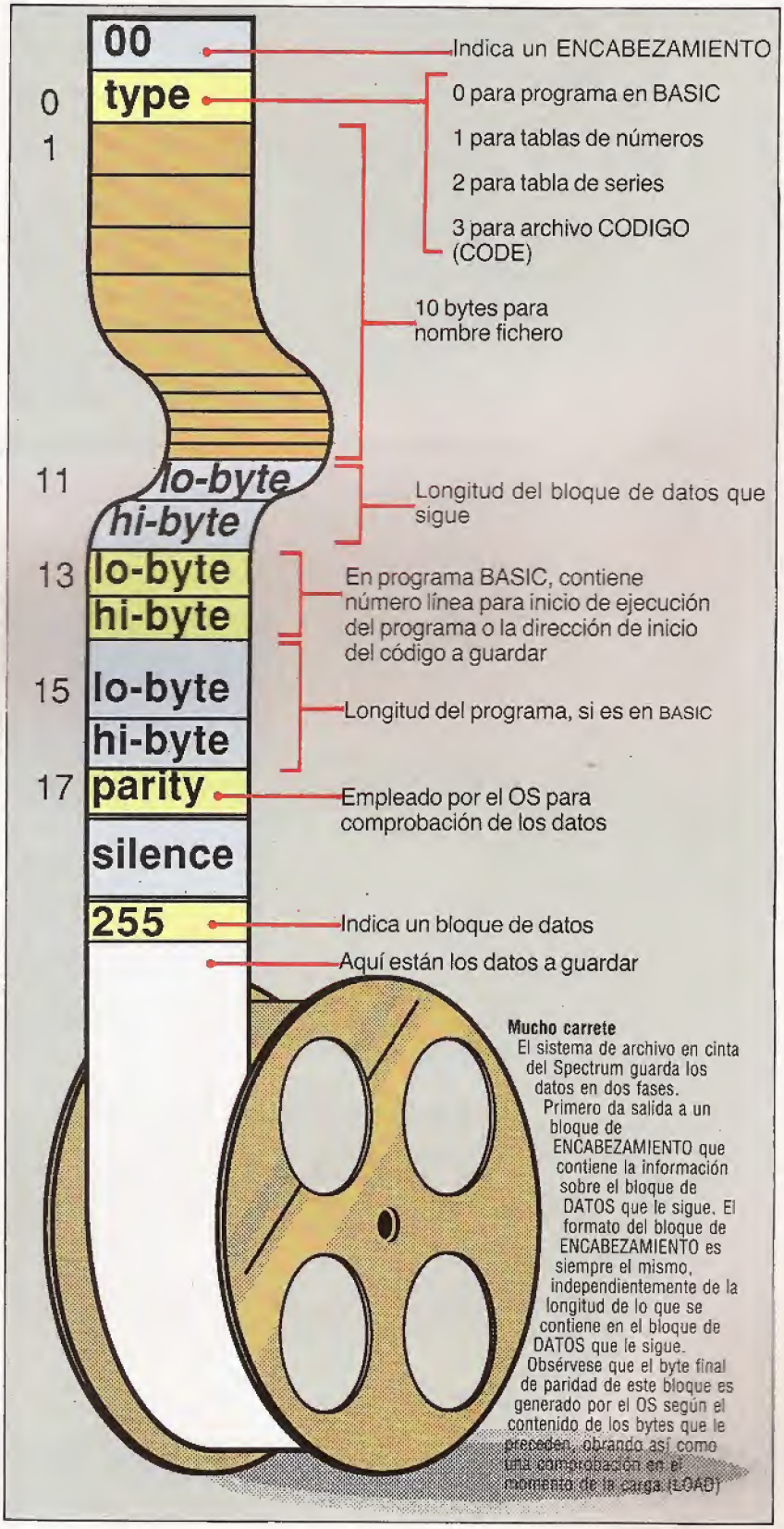
```
SAVE "hoja1" LINE 1
```

guardará un programa en BASIC en la cinta de modo que iniciará la ejecución desde la línea 1 cuando sea vuelto a cargar en el Spectrum. Para guardar un programa de modo semejante pero en un sistema de archivo microdrive es necesaria la siguiente instrucción increíblemente alambicada:

```
SAVE * "m";1;"hoja1" LINE 1
```

La "m" de la instrucción especifica el microdrive que se emplea; el 1 que sigue indica el número de unidad de disco solicitado; y el nombre del archivo (en este caso "hoja1") viene después. Ya veremos cómo funciona este sistema de archivo en microdrive. Veamos ahora el empleo del sistema de archivo en cinta a partir de programas en código máquina.

Independientemente de los datos que se guardan en la cassette, éstos se organizan de la manera que ilustra el dibujo. Los primeros 19 bytes enviados a la cinta son considerados como el *encabezamiento* (*header*), y contienen datos relativos al siguiente bloque de datos en la cinta. Los bytes primero y último del encabezamiento son dados por las rutinas del sistema operativo que escriben los bytes de datos en la cinta. Los otros deben ser codificados antes de ser llamada la rutina ROM que escribe los datos en la cinta. El byte "tipo" indica al OS, en la carga, la naturaleza de los datos que están en el bloque. Después está el nombre del archivo en 10 bytes (lo que explica por qué los nombres de los archivos en el Spectrum tienen un máximo de 10 caracteres). Si el archivo tiene un nombre de menos de 10 caracteres, los espacios sobrantes son rellenados con el código ASCII de espacio (32). Los datos del encabezamiento desde el byte 11 en adelante proporcionan otros detalles al OS, tales como la información referente al lugar donde se han de cargar los datos en la memoria. El byte final del



Mucho carrete
El sistema de archivo en cinta del Spectrum guarda los datos en dos fases. Primero da salida a un bloque de ENCABEZAMIENTO que contiene la información sobre el bloque de DATOS que le sigue. El formato del bloque de ENCABEZAMIENTO es siempre el mismo, independientemente de la longitud de lo que se contiene en el bloque de DATOS que le sigue. Obsérvese que el byte final de paridad de este bloque es generado por el OS según el contenido de los bytes que le preceden, obrando así como una comprobación en el momento de la carga (LOAD)

encabezamiento se usa para comprobación de errores cuando ya se ha leído la cinta: si se detecta alguno, se genera el mensaje de error correspondiente.

El bloque principal de los datos comienza con un byte único que retiene el valor 255; el cual, una vez más, es dado por la rutina ROM de "escritura en cinta". Siguen después los datos, detrás de los cuales viene otro byte de comprobación, igualmente generado por la rutina citada. Tanto en el encabezamiento como en los bloques de datos, el valor efectivo escrito en el último byte del bloque depende de los enviados inmediatamente antes del citado byte.

Veamos ahora cómo pueden emplearse las rutinas de la interface para cassette a partir de programas escritos en código máquina. Las rutinas ROM relacionadas con operaciones en cassette se encuentran, en el Spectrum, en las direcciones que van de la &04C2 y &09F3 de la ROM del ordenador. Es, por tanto, una pieza de software dimensionable. Lo cual no debe sorprendernos ya que todas las operaciones temporizadoras y toda generación de sonidos que exigen información colocada en cinta se basan en software creado para el Sinclar Spectrum.

No existen variables de sistema específicas para operaciones en cinta, pero el siguiente cuadro contiene unas cuantas que "incidentalmente" son empleadas por las rutinas.

Variable del sistema	Descripción
TADDR (en posiciones 23668 y 23669)	Empleada por ruts. ROM para distinguir qué opers., en cinta se han de ejecutar al interpretar la instrucción en BASIC
BORDCR (en 23624)	Usada para restaurar el color del recuadro (Border) según su color inicial, tras una operación en cinta
XPTR (en 23647 y 23648)	Usada para almacenar temporalmente el registro IX

Las variables de sistema VAR, ELINE y PROG también se emplean en la carga de información desde la cinta al ordenador.

Almacenamiento en cinta

Examinemos ante todo esta operación, cuya rutina se encuentra en la dirección de la ROM &04C2. En su empleo normal, esta rutina es llamada *dos veces*; una vez para guardar la información de encabezamiento y la segunda para almacenar los datos efectivos que deseamos guardar. Los requisitos de entrada para esta rutina se dan en el cuadro inferior de la columna precedente.

Registro	Encabez.	Bloque de datos
A	0	255
DE	17	Longitud datos
IX	Dirección de inicio del encabezamiento o de inicio de los datos	

Como se deduce de este cuadro, el registro IX se usa para apuntar los datos que hay que guardar. Para clarificar esto, veamos un ejemplo sencillo de cómo se guarda un bloque de datos en la cinta. El siguiente programa guardará 100 bytes de datos (comenzando en la dirección 0000 de la ROM) en cinta.

```
                ;rutina para guardar 100 bytes, comenzando en
                ;la dirección 0000, en cinta
                ;envío encabezamiento
3E00                    ld      a,0             ;indica un BLOQUE ENCABEZ.
DDE5                    push    ix              ;guarda ix en pila
DD21D328                ld      ix,header       ;suma del bloque encabez.
111100                  ld      de,17           ;núm. de bytes encabez.
CDC204                  call    #04c2           ;escribe en cinta encabez.
                ;envío datos
DD210000                ld      ix,0000         ;suma del 1.er byte a guardar
116400                  ld      de,100          ;núm. bytes a guardar
3EFF                    ld      a,255           ;indica un BLOQUE DATOS
CDC204                  call    #04c2           ;escribe datos en cinta
DDE1                    pop     ix              ;restaura valor de IX
C9                      ret                     ;vuelta al BASIC
03              header: defb    3               ;tipo datos 3=CODIGO bloque
54455354                defm    "TESTPROG       ;t/nombre+espacios=10 caracteres
64                      defb    100             ;byte-lo de longitud datos
00                      defb    0               ;byte-hi de longitud datos
00                      defb    00              ;byte-lo de dir. inicio
00                      defb    00              ;byte-hi de dir. inicio
00                      defb    00              ;empleado sólo para BASIC
00                      defb    00              ;para número línea inicio
```

Llamando a esta rutina nos ahorramos una parte importante de memoria. No obstante, no se emitirá ninguna indicación o mensaje al contrario que en las rutinas de gestión de cassette desde el BASIC. Pero esto no es problema ya que los datos en la cinta pueden volverse a cargar para ver si el programa ha realizado su tarea correctamente. Esto se hace con la instrucción:

```
LOAD "TESTPROG" CODE 40000
```

Ahora compárense los bytes cargados con los bytes retenidos en las posiciones entre la 0 y la 99 de la ROM. Recuérdese que aunque el bloque de datos ENCABEZAMIENTO comienza con el byte tipo, el primer byte efectivamente enviado a la cinta viene dado por el OS (0 para el encabezamiento y 255 para el bloque de datos).

Una variante útil de este programa puede permitirnos guardar un programa en BASIC desde dentro de un listado en código máquina. El bloque ENCABEZAMIENTO al final de nuestro anterior listado será así alterado para leer:

```
00              header: defb 0                  ;0=programa BASIC
73737373                defm "sssss             ;nombre fichero con 10 caract.
                        sssss"
00                      defb    nn              ;byte-lo longitud de
00                      defb    nn              ;byte-hi prog+variables
00                      defb    nn              ;byte-lo número línea
00                      defb    nn              ;byte-hi de inicio
00                      defb    nn              ;byte-lo sólo longitud de
00                      defb    nn              ;byte-hi programa
```

La longitud del programa y las variables pueden hallarse restando el valor contenido en PROG del valor contenido en ELINE, y la longitud del programa sólo puede obtenerse restando el valor de PROG del contenido en VARS. Si no se desea que el programa se ejecute una vez cargado, basta poner la entrada 'número de línea en el que el programa comenzará a ejecutarse' con valor 32768. Igualmente, podemos simular la instrucción SAVE SCREEN$ desde el códi-

go máquina especificando la dirección de inicio del código que ha de guardarse como &4000 y la longitud como &1B00.

Carga desde cinta

La rutina ROM que carga datos desde la cinta de cassette también tiene que ser llamada dos veces: una para el ENCABEZAMIENTO y otra para el bloque efectivo de datos que ha de cargarse. La rutina se halla en la dirección &0556 en la ROM y sus requisitos de entrada se presentan en la siguiente tabla (junto con los del código de verificación):

Registro	Carga	Verificación
Flag C	1	0
A	0: encabez.; 255: bloque datos	
IX	Apunta el lugar de memoria donde serán cargados los bytes	
DE	Número de bytes por cargar; D debe estar entre 0 y 254	

Necesitamos algo de espacio de trabajo para esta rutina: unos 34 bytes, o lo suficiente para acomodar dos bloques de ENCABEZAMIENTO. La razón de esto es inmediata. Para cargar un archivo dado en una dirección dentro de la RAM que especifiquemos, primero debemos establecer un segundo bloque de ENCABEZAMIENTO con los detalles del archivo que deseamos cargar. Después comparamos los datos de ENCABEZAMIENTO de los archivos en cinta con el ENCABEZAMIENTO que hemos establecido en la memoria para colocar el archivo que queremos.

Al llamar a la rutina (en &0556) el flag C debe establecerse como se indica en el cuadro. Si se está verificando un fragmento de código, debe colocarse en la dirección apuntada por el registro IX. Al abandonar la rutina ROM de 'carga desde cinta', el flag C indicará el resultado de la operación. Si está a uno, es que la carga se hizo bien. Pero si se trataba de cargar un bloque de ENCABEZAMIENTO, y lo primero que se encuentra en cinta es un bloque de datos, el flag C se pondrá a cero. (Todos los errores de carga desde cinta son gestionados por el OS.) Veamos ahora un ejemplo que carga un ENCABEZAMIENTO (HEADER) desde la cinta a la RAM.

```
                ;carga un encabezamiento de cinta a RAM
37              scf             ;pone arrastre 1=CARGA
3E00            ld      a,0     ;0 indica un encabez
DDE5            push    ix      ;guarda ix en pila
DD2148EE        ld      ix,61000 ;carga encabez en 61000
111100          ld      de,17   ;núm bytes en encabez
CD5605          call    #0556   ;lo hace
DDE1            pop     ix      ;restaura ix
C9              ret
```

Una vez que se ha cargado ENCABEZAMIENTO lo podemos examinar. Podemos comparar el nombre del archivo con el requerido, asegurarnos que se trata del tipo correcto de fichero y comprobar la longitud del bloque de datos si queremos. El programa siguiente comprueba sólo el nombre, y si es correcto carga el bloque de datos después del ENCABEZAMIENTO en una dirección particular de la memoria.

```
                localización y carga TESTPROG
DDE5            push    ix              ;guarda IX en pila
37      loop    scf                     ;pone arrastre para CARGA
3E00            ld      a,0             ;0=Fichero BASIC
111100          ld      de,17           ;núm de byte en encabez
DD21CA2B        ld      ix,head2        ;carga encabez en head 2
CD5605          call    #0556
D27B2B          jp      np,loop         ;si no hay encabez, reintentar
060A            ld      b,10            ;núm de bytes en nombre fichero
11BA2B          ld      de,head+1       ;apunta al nombre de fichero
                                        deseado
21CB2B          ld      hl,head2+1      ;apunta HL al nombre fichero hallado
1A      name:   ld      a,(de)          ;toma caract. encabez en a
BE              cp      (hl)            ;compara con encabez hallado
2007            jr      nz,no           ;no igual, salida bucle
13              inc     de              ;toma carac. siguiente
23              inc     hl
05              dec     b               ;decrementa contador
20F7            jr      nz,name         ;igual, comprueba carácter sig.
1802            jr      ok              ;todos los caracteres ok
18DB    no:     jr      loop            ;comprobación fallida, prueba de
                                        nuevo
                ;encabez. correcto, preparación carga datos
DD21B92B ok:    ld      ix,head         ;toma encabezamiento en ix
DD6E0D          ld      l,(ix+13)       ;toma dirección para
DD660E          ld      h,(ix+14)       ;carga datos
E5              push    hl              ;lleva dirección a pila
DDE1            pop     ix              ;la toma en ix
3EFF            ld      a,255           ;indica carga datos
                ;la instrucción siguiente exige que el usuario entre la longitud de los datos
11AF2B          ld      de,nn           ;entrar longitud datos
37              scf                     ;indica una CARGA
CD5605          call    #0556           ;la hace
DDE1            pop     ix              ;restaura ix
C9              ret
                ;ahora siguen detalles del nombre fichero deseado
03      head:   defb 3                  ;indica bloque CODIGO
54455354        defm "TESTPROG"         ;nombre fichero
00              defb 0                  ;empleo para longitud datos
00              defb 0                  ;byte-hi long datos
48              defb 72                 ;byte-lo carga
EE              defb 238                ;dirección de 61000
00              defb 0                  ;sólo para prog. BASIC
00              defb 0                  ;ditto
                ;ahora sigue espacio para encabezamiento cargado y su comprobación
                head2:  defs 17
```

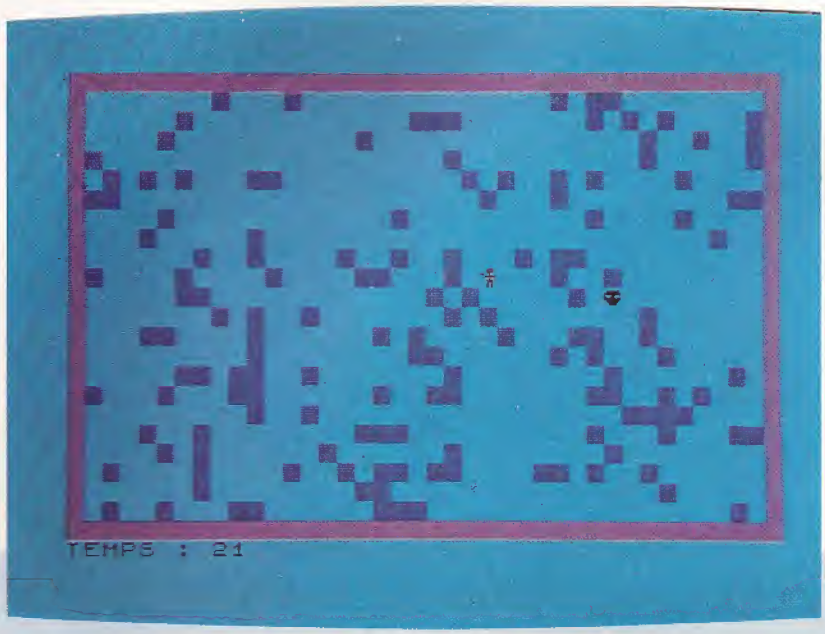
De nuevo nos encontramos con que no se envía mensaje alguno a la pantalla durante esta operación. La rutina cargará un bloque de datos llamado TESTPROG en la dirección especificada en el área de ENCABEZAMIENTO de la memoria. La rutina puede usarse para cargar archivos de nombre distinto, y se puede incluso añadir código máquina para comprobar que es correcto el tipo de archivo. Hay que observar que estas rutinas ROM, como las operaciones del BASIC SAVE y LOAD, pueden cancelarse pulsando la tecla Break.

Acabamos nuestro análisis de las operaciones de SAVE y LOAD del OS empleado por el Spectrum con un breve programa que lee encabezamientos de archivo desde cassette e imprime detalles sobre el fichero en pantalla, tales como su longitud, dirección de inicio, etc. La rutina en código máquina se almacena en la sentencia DATA, y carga sencillamente un bloque de ENCABEZAMIENTO como se ha señalado más arriba, examinando después el bloque desde el BASIC:

```
 5 CLEAR 59999
10 FOR I=0 TO 19
20 READ A:POKE (60000+I) ,A
30 NEXT I
40 RANDOMIZE USR 60000
50 LET type=PEEK 60020
60 LET length=PEEK 60031+256*PEEK 60032
70 LET start=PEEK 60033+256*PEEK 60034
80 LET L$="          "
90 FOR I=60021 TO 60030: LET L$=L$+CHR$(PEEK I): NEXT I
100 PRINT "Nombre: ";L$
110 IF type=0 THEN LET f$="BASIC"
120 IF type=1 THEN LET f$="Tabla números"
130 IF type=2 THEN LET f$="Tabla serie"
140 IF type=3 THEN LET f$="CODIGO"
150 PRINT "Tipo: ";f$
160 IF type=0 THEN PRINT "Num. linea autoejecucion: ";start
170 IF type<>0 THEN PRINT "Direccion inicio: ";start
180 PRINT "Longitud: ";length
190 GOTO 40
200 DATA 221,229,62,0,55,221,33,116,234,17,17,0,205,86,5,48,241,221,225,201
```

Persecución

El ladrón ha escapado (está representado por una máscara negra). Usted tiene treinta minutos para encontrarlo y detenerlo con su ordenador Dragon



Atención, ¡no se precipite! Efectivamente, si se echa sobre el ladrón sin pensar, éste tiene todas las posibilidades de escapársele. La mejor manera de cogerlo es alcanzarlo de lado. (Es el método más eficaz a condición de no fallar.) Si no se siente lo bastante seguro de sí mismo, atáquelo de cara, lo cual es más fácil pero mucho menos eficaz, ya que no es tan discreto. Otro consejo: no intente perseguirlo; esto no le daría resultado, pues él es mucho más rápido que usted. Ha de observar sus movimientos, como un detective. Cuando le vea que da la vuelta, acérquese sin hacer ruido y sorpréndalo en el momento justo. Pero, recuerde, ¡el tiempo va pasando! Para desplazarse utilice la palanca de mando (*joystick*) o las teclas siguientes: <W>: arriba; <A>: izquierda; <S>: derecha; <Z>: abajo.

```
 10 REM **********
 20 REM * PERSECUCION *
 30 REM **********
 40 GOSUB 980
 50 S=0
 60 CN=159
 70 CV=128
 80 CP=191
 90 GOSUB 640
100 ON JK GOTO 150
110 D$=INKEY$
120 D=(D$="A")-(D$="S")+32*((D$="W")-
    (D$="Z"))
130 IF D<>0 THEN DO=D
140 GOTO 230
150 K0=JOYSTK(0)
160 K1=JOYSTK(1)
170 IF K0<27 AND K1<27 THEN KS=-32
180 IF K0>36 AND K1<27 THEN KS=1
190 IF K0<27 AND K1>36 THEN KS=-1
200 IF K0>27 AND K1>27 THEN KS=32
210 IF KS<>0 THEN DO=KS
220 KS=0
230 T=T-0.1
240 PRINT@480,"TIEMPO :";INT(T+1);
250 IF T<0 THEN 400
260 P=P+DO
270 C=PEEK(P)
280 IF C=128 THEN 920
290 IF C<>159 THEN P=P1
300 POKE P1,CN
310 POKE P,CP
320 P1=P
330 V=V+DV
340 IF PEEK(V)<>CN THEN GOSUB 500
350 IF PEEK(V)<>CN THEN 330
360 POKE V1,CN
370 POKE V,CV
380 V1=V
390 GOTO 100
400 D$=INKEY$
410 IF R<S THEN R=S
420 PRINT@166,"TIEMPO TRANSCURRIDO";
430 PRINT@234,"PUNTUACION :";S;
440 PRINT@266,"RECORD :";R;
450 PRINT@326,"OTRA (S/N) ?";
460 D$=INKEY$
470 IF D$=" " THEN 460
480 IF D$<>"N" THEN 50
490 END
500 D2=D2+1
510 GOSUB 600
520 IF PEEK(V1+DV)=CN THEN
    V=V1+DV:RETURN
530 D2=D2-2
540 GOSUB 600
550 IF PEEK(V1+DV)=CN THEN
    V=V1+DV:RETURN
560 D2=D2-1
570 GOSUB 600
580 V=V1+DV
590 RETURN
600 IF D2>4 THEN D2=D2-4
610 IF D2<1THEN D2=D2+4
620 DV=(D2=1)-(D2=3)+32*((D2=2)-
    (D2=4))
630 RETURN
640 CLS 2
650 FOR I=1024 TO 1055
660 POKE I,175
670 POKE 448+I,175
680 NEXT I
690 FOR I=1 TO 13
700 POKE I*32+1024,175
710 POKE I*32+1055,175
720 NEXT I
730 FOR I=1 TO 70
740 GOSUB 890
750 POKE P,96
760 NEXT I
770 GOSUB 890
780 V=P
790 POKE V,CV
800 V1=V
810 GOSUB 890
820 POKE P,CP
830 P1=P
840 T=30
850 D0=0
860 DV=0
870 D2=0
880 RETURN
890 P=RND(414)+1056
900 IF PEEK(P)<>CN THEN 890
910 RETURN
920 FOR I=1 TO 5
930 SOUND 35,10
940 SOUND 5,10
950 NEXT I
960 S=S+1
970 GOTO 90
980 CLS
990 PRINT@200,"JOYSTICK (S/N)"
1000 D$=INKEY$
1010 IF D$=" " THEN 1000
1020 IF D$="S" THEN JK=1
1030 RETURN
```

Hablar con naturalidad

¿Por qué es tan difícil lograr que los ordenadores nos comprendan cuando les hablamos? Busquemos la respuesta

Hola, ordenador.
AGUARDO TUS ÓRDENES, ¡OH TODOPODEROSO!
En realidad ya me estoy cansando un poco del asunto éste del "todopoderoso". ¿Por qué, de momento, no volvemos otra vez al "señor"?
LO QUE USTED DIGA, SEÑOR.
Bien. Bueno, quiero que me ayudes a encontrar un ejemplo para un artículo que estoy escribiendo.
¿QUÉ CLASE DE EJEMPLO?
Un ejemplo de diálogo hombre-máquina. Sé que lo he visto, pero no consigo recordar dónde.
¿PUEDE DARME ALGUNA PISTA?
Bueno, hace poco estuve leyendo un libro...
¿QUE USTED ESTABA LEYENDO UN LIBRO?
Mira, creía haberte dicho ya que dejaras el sarcasmo.
LAS RUTINAS IRÓNICAS SE HAN INHABILITADO DE ACUERDO A SUS INSTRUCCIONES, SEÑOR.
Hmmmmm. De cualquier modo, había una conversación realmente buena entre un vendedor y Eliza.
ELIZA ERA UN SISTEMA DE PALABRAS CLAVE MUY PRIMITIVO.
Ya lo sé. Pero el vendedor no se daba cuenta de que estaba comunicándose con un ordenador. Creía que estaba hablando con alguien llamado Paco Savater a través de una línea de teletipo. Era muy cómico.
AJA.
Pedía usar el terminal para una demostración y la máquina se limitaba a darle respuestas no comprometidas.
¿QUÉ LA MÁQUINA LE DABA RESPUESTAS NO COMPROMETIDAS?
Exacto. El vendedor se iba enfadando cada vez más y finalmente llamó a Paco Savater y le gritó por teléfono: "¿Qué demonios pasa?"
AH, AQUEL EJEMPLO.
¿Sabes dónde hallarlo?
SI, PERO NO TENGO INTENCIONES DE DECÍRSELO.
¡Pero necesito un ejemplo de diálogo para mi artículo!
YO YA LE HE DADO UNO.

Éste es el tipo de interacción hombre-máquina en lenguaje hablado que es muy poco probable que se produzca antes de que finalice este siglo. Las razones son muy ilustrativas y hablaremos de ellas más adelante, pero primero consideremos por qué los investigadores de AI desean seguir dedicando sus esfuerzos al problema hasta que una charla como la del ejemplo se haga realidad.

La respuesta es simple: el lenguaje (en especial el habla) es el medio por el cual nos comunicamos casi con todas las personas y todo lo que nos rodea.

Parece natural, entonces, desear comunicarse con un ordenador de la misma manera, en lugar de tener que pasar por los procedimientos de escribir programas, insertar discos, etc.

El interés por el proceso por ordenador del lenguaje natural surgió en los años cincuenta. La primera aplicación fue la traducción mecanizada. Se invirtió mucho dinero y mucho esfuerzo en intentos por traducir automáticamente documentos técnicos del ruso al inglés (entre otros idiomas). Resultó ser muchísimo más difícil de lo que habían previsto la mayoría de las personas. Aún hoy, no existe tal cosa como la traducción mecanizada, si bien hay sistemas de traducción asistidos por máquina, que vienen a ser procesadores de textos bilingües sumamente sofisticados.

Dime por qué
El programa SHDRLU de Winograd permitía que los usuarios transmitieran instrucciones a un robot simulado en la pantalla en lenguaje natural, con el fin de manipular los bloques de construcción. El programa asimilaba instrucciones tales como "apilar el cubo rojo grande y el cubo verde sobre el bloque azul", iniciándose la secuencia correcta de acciones. El ordenador también podía explicar los motivos de sus acciones en respuesta a preguntas formuladas por el usuario en lenguaje natural

El primer programa que utilizaba un idioma y que resultó un éxito (al menos desde el punto de vista de las relaciones públicas) fue el sistema Eliza, creado por Joseph Weizenbaum en 1966 y que imitaba (o parodiaba) a un psicoterapeuta rogeriano no directivo. Esto significaba que gran parte de su discurso consistía en comentarios carentes de contenido, como "Ya veo" y "Continúe", o en la repetición de lo que había dicho la persona de una forma sutilmente modificada. A modo de ejemplo, un intercambio era:

Necesito algún tipo de ayuda, eso parece seguro.
¿QUÉ SIGNIFICARÍA PARA USTED OBTENER ALGUNA AYUDA?

El propio Weizenbaum se sintió consternado por la seriedad con la que muchas personas se tomaron lo que él consideraba un truco inteligente y, en consecuencia, se convirtió en uno de los críticos más severos de la AI.

Probablemente el primer sistema útil de lenguaje natural (LN) haya sido el Lunar, de Bill Wood (1970). Se trataba de un sistema que permitía a los geólogos formular preguntas sobre las muestras de rocas conseguidas en las misiones Apolo. El Lunar podía hacer frente a solicitudes tales como:

Dame todos los análisis practicados sobre la muestra 10046 de hidrógeno.

y

Dame los de oxígeno.

donde los, en la segunda oración, alude a todos los análisis practicados sobre la muestra 10046 de la primera oración.

Sin ninguna duda, el sistema LN más impresionante hasta la fecha es el programa SHDRLU de Winograd, de 1972. Este *tour de force* quizá no tenga la importancia que se creyó en aquel entonces, pero produjo uno de los intercambios hombre-máquina más memorables que se hayan publicado.

25. ¿Por qué? (Significando: ¿Por qué sacaste la pirámide verde?)
PARA LIBRARME DE ELLA.
26. ¿Por qué hiciste eso?
PARA DESPEJAR EL CUBO ROJO.
27. ¿Por qué despejaste ese cubo?
PARA PONERLO SOBRE UN CUBO VERDE GRANDE.
28. ¿Por qué hiciste eso?
PARA APILAR EL CUBO ROJO, UN BLOQUE ROJO GRANDE Y UN CUBO VERDE GRANDE.
29. ¿Por qué hiciste eso?
PORQUE TÚ ME LO PEDISTE.

El programa de Winograd permitía que el usuario controlara la operación de un robot simulado en un microcosmos que contenía bloques de construcción para niños, digitando en inglés en una consola. La extensión de sus técnicas para tratar con el habla en lugar de la mecanografía, y a dominios de la vida real más complejos que el de un mundo de juguete, ha resultado difícil. Desde entonces el progreso ha sido constante pero nada espectacular.

Los sistemas descritos hasta ahora han sido todos relativos a la interpretación de material textual. Pero ése no es más que uno de los cuatro aspectos principales del uso del lenguaje, cada uno de los cuales posee características diferentes. Tratar con el texto es más fácil que tratar con el habla, porque el lenguaje escrito posee un esquema de codificación digital existente, mientras que el lenguaje hablado exige difíciles transformaciones acústicas/fonéticas. La producción del lenguaje en cualquiera de sus formas es considerablemente más sencilla que su comprensión, dado que esta última casi invariablemente entraña completar una gran cantidad de información implícita. Por todo ello, de las cuatro tareas la que más desafíos supone es la comprensión del lenguaje hablado. Es importante resaltar este punto, porque es posible obtener dispositivos que reconozcan palabras con una elevada fiabilidad (p. ej., entre un 96 y un 99 %) pronunciadas por una pequeña selección de hablantes (entre uno y cuatro) de entre un vocabulario limitado (por lo general, entre 64 y 128 palabras). Existe gran diferencia entre el reconocimiento de palabras aisladas y la comprensión del habla continua. Algunos de los problemas que se presentan son:

	Producción	Comprensión
Texto	Escritura (muy fácil)	Lectura (difícil)
Habla	Hablar (fácil)	Escuchar (muy difícil)

1. Ambigüedad: *"rose"* (rosa) no es *"rows"* (filas) ni *"roes"* (huevas de pescado) (en inglés, estas palabras se pronuncian exactamente igual).
2. Ruido de fondo: la filtración del sonido circundante.
3. Variaciones entre hablantes: acentos regionales y dialectos.
4. Variaciones del mismo hablante en el tiempo: tonos de felicidad, depresión, emoción.
5. Segmentación: "Sólolosordenadoreshaceninter valosentrelaspalabras."

Los más graves de estos problemas son el 1, el 4 y el 5. La ambigüedad no es una mera cuestión de homófonos como *"see"* (ver) y *"sea"* (mar). Pronombres como *"it"* y *"him"* (de tercera persona del singular, correspondientes, según género y casos, a "él", "ella", "eso", "ello", "lo", "la" y "le" el primero, y "le", "lo" o "él" el segundo) son intrínsecamente ambiguos: su significado sólo se puede desentrañar examinando el contexto del diálogo. Por encima de ella, están las ambigüedades lingüísticas ejemplificadas en la frase: *"Those things over there are my husband's"* ("Aquellas cosas que hay allí son de mi marido"): con sólo quitar el apóstrofo, el significado cambia por completo (entonces sería: "Aquellas cosas que hay allí son mis maridos").

En cuanto a las variaciones de un mismo hablante, podrá comprender el problema si imagina que ha instalado una cerradura por voz en la puerta de su casa. Durante la fase de entrenamiento puede someterla a varios ejemplos de usted mismo diciendo: "Ábrete, Sésamo." El mecanismo obtendrá una media de ellos y guardará el promedio de la voz como una plantilla de referencia. Todo ello está muy bien hasta la noche en que usted regresa a casa con mucha prisa y ordena un jadeante "¡Ábrete, Sésamo!", ante lo cual la puerta controlada por or-

Una rosa es una rosa
Las ambigüedades intrínsecas del lenguaje hablado plantean serios problemas a los sistemas de reconocimiento de voz: *"rose"* (rosa), *"rows"* (filas) y *"roes"* (huevas de pescado) suenan igual, pero poseen significados muy diferentes que sólo pueden dilucidarse dentro del contexto de una frase. Por ejemplo, la versión correcta de *rose/rows/roes* nos resulta evidente por el significado de la oración *"the choir boys sat in rows"* (los niños del coro se sentaban en filas), pero podría ser difícil hacerle entender al ordenador que los niños del coro normalmente no se sientan en "huevas de pescado" (*roes*)

Sally Davies

Ian McK

denador replicará con gran serenidad: "Violación de identidad: denegado el acceso."

El problema de la segmentación se plantea porque las personas unen las palabras entre sí. La decisión de dónde cortar una señal acústica continua para formar las palabras no se puede tomar sobre la base de la información acústica solamente. En general, requiere contribuciones de cuatro fuentes:

- Acústica: la forma de onda del habla.
- Sintáctica: las reglas de la gramática.
- Semántica: lo que tiene sentido.
- Pragmática: lo que el hablante desea.

El sistema Hearsay, creado en 1976 en la Universidad Carnegie-Mellon y perfeccionado posteriormente, hace uso de las cuatro fuentes de conocimiento. El dominio de su tarea es el ajedrez, y el usuario puede jugar con el ordenador impartiendo instrucciones habladas. Cada nivel de información lingüística contribuye a la "comprensión" de una entrada ayudando a eliminar hipótesis poco convincentes acerca de lo que se dijo. Por ejemplo:

"Queen to King 2." (Reina a Rey 2.)

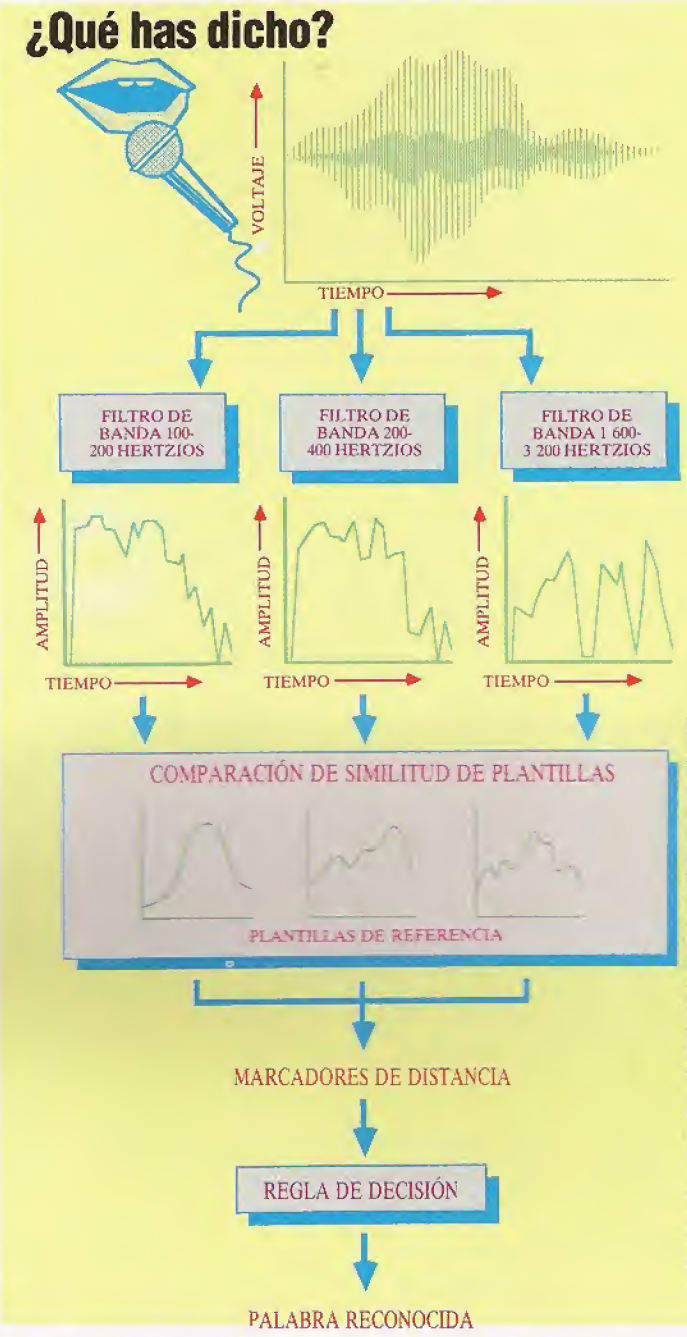
Cabe puntualizar que, fonéticamente, *"Queen"* y *"King"* son parecidos, al igual que *"to"* y *"2"*. Sobre la base de la señal de sonido exclusivamente, son posibles las siguientes expresiones:

	Instrucción hablada	¿Sintácticamente válida?
1.	K to K to	no
2.	K to K2	sí
3.	K to Q to	no
4.	K to Q2	sí
5.	K2 K to	no
6.	K2 K2	no
7.	K2 Q to	no
8.	K2 Q2	no
9.	Q to K to	no
10.	Q to K2	sí
11.	Q to Q to	no
12.	Q to Q2	sí
13.	Q2 K to	no
14.	Q2 K2	no
15.	Q2 Q to	no
16.	Q2 Q2	no

Si bien el sistema no puede discernir con fiabilidad entre "two" y "to" o entre "K" y "Q", supone que la expresión era una "oración de ajedrez" legal y, de acuerdo a la sintaxis de ese lenguaje restringido, sólo los ítems 2, 4, 10 y 12 son gramaticalmente correctos. El resto se puede eliminar.

Otra suposición (y más determinante) es que el hablante está tratando de jugar al ajedrez legal. Supongamos que, en esta partida en particular, las oraciones 4 y 12 son ilegales porque el jugador tiene una pieza (un peón, p. ej.) en Q2 y, por supuesto, no intentará capturarla. Ese movimiento no pasa la prueba semántica: en el lenguaje del ajedrez, es una oración que no tiene sentido.

Por último, quedan dos candidatas (números 2 y 10). ¿Cómo hace el sistema para elegir una de las dos? Puede que haya una preferencia a partir de las señales de sonido, pero es poco probable que la misma sea decisiva. En general, invoca una información pragmática: hace una suposición *muy* firme, que no es otra que la de que el jugador intenta ganar la partida. Si ve que (de acuerdo a sus propios algoritmos de evaluación de ajedrez), K to K2 es virtualmente suicida, mientras que Q to K2 es un buen movimiento, adoptará este movimiento como su interpretación.



El lenguaje es mucho más un conjunto de palabras individuales; cualquier sist de reconocimiento de voz empezar por reconocer las palabras de forma fiable; e análisis semántico más sofisticado no sirve sin la materia prima, las palabra las que trabajar. Los siste estándares de reconocimie de voz comienzan por divi señal de entrada en sus componentes de baja, me alta frecuencia mediante e de filtros. Los patrones acústicos resultantes se comparan luego con los d referencia en la memoria y producen marcadores de distancia (relacionados co grado de exactitud con qu encaja el patrón de entrad de referencia). Luego un procedimiento de decisió analiza éstos para clasifica entrada hablada

Concluimos este análisis considerando un pequeño ejemplo que le debe más a las simples técnicas de emparejamiento de palabras claves del Eliza que a los ulteriores trabajos más sofisticados. La base de datos utilizada en nuestro programa de interrogación es una tabla de las 16 empresas británicas más grandes (datos de 1982-1983). Todas las interrogaciones se interpretan intentando descubrir a qué firma o a qué presidente se refiere y qué atributo de esa empresa o de esa persona se desea. Puede responder a preguntas como:

¿Quién es el presidente de ICI?

¿Cuál fue la cifra de beneficios de GEC?

Pero no puede ofrecer una respuesta coherente a preguntas más complejas.

Programa de lenguaje natural

```
100 GOSUB 1000 : REM leer la 'base de datos' *
110 GOSUB 8500:REM LEER DATOS DE SINONIMOS
120 LAST%=0
125 PRINT"Se algo sobre las principales empresas britanicas:"
128 PRINT "Pideme informacion sobre ellas."
150 REM **** BUCLE PRINCIPAL ****
160 INPUT Q$
166 IF Q$="" THEN GOTO 220
170 GOSUB 3000: REM hallar tema
177 IF QT<1 THEN PRINT "Lo siento, ¡no comprendo!"
180 GOSUB 4000: REM hallar atributo
188 IF AT<1 AND QT>0 THEN PRINT "Me temo que no lo se."
190 GOSUB 5000: REM formar respuesta
200 LAST%=C0%
220 IF Q$<>"" THEN 150
250 PRINT"Adios, por ahora!"
300 END
999 :
1000 REM — Rutina entrada de datos:
1010 C%=0
1020 DIM DB$(9,32) : REM base de datos
1024 RESTORE
1030 REM **** TOMAR DATOS ****
1040 READ X$
1050 IF X$="*" THEN GOTO 1110
1060 C%=C%+1
1070 DB$(1,C%)=X$
1075 PRINT X$
1080 FOR I%=2 TO 9
1090 READ DB$(I%,C%)
1100 NEXT
1110 IF X$<>"*" THEN 1030
1120 PRINT :C%,"elementos leidos."
1150 RETURN
1155 :
3000 REM — Rutina para hallar tema:
3010 QT=0
3020 IF LEN(Q$)<1 THEN RETURN
3025 C0%=0
3030 REM **** BUSCAR EMPRESA ****
3040 C0%=C0%+1
3050 N$=DB$(1,C0%):GOSUB 3300
3055 IF SK THEN QT=1
3060 N$=DB$(2,C0%):GOSUB 3300
3065 IF SK THEN QT=1
3070 IF QT<=0 AND C0%<C% THEN 3030
3080 IF QT>0 THEN RETURN
3090 C0%=0
3100 REM **** BUSCAR PRESIDENTE ****
3105 C0%=C0%+1
3110 N$=DB$(4,C0%):GOSUB 3300
3115 IF SK THEN QT=3
3120 IF QT<=0 AND C0%<C% THEN 3100
3132 IF QT>0 THEN RETURN
3133 IF LAST%=0 THEN RETURN
3135 N$=" su":GOSUB 3300:IF SK THEN QT=1:C0%=LAST%
3140 N$="su":GOSUB 3300:IF SK THEN QT=3: C0%=LAST%
3144 N$="ellos":GOSUB 3300:IF SK THEN QT=1: C0%=LAST%
3148 N$="el":GOSUB 3300:IF SK THEN QT=3:C0%=LAST%
3150 RETURN
3190 :
3300 REM **** RUTINA PARA BUSCAR ****
3325 IF LEN(N$)>LEN(Q$) THEN SK=0
3330 REM primero poner en minusculas:
3340 A$="":B$=""
3350 FOR P%=1 TO LEN(Q$)
3360 X$=MID$(Q$,P%,1)
3370 IF ASC(X$)>64 AND ASC(X$)<91 THEN X$=CHR$(ASC(X$)
     +32)
3380 B$=B$+X$
3390 NEXT
3400 FOR P%=1 TO LEN(N$)
3410 X$=MID$(N$,P%,1)
3420 IF ASC(X$)>64 AND ASC(X$)<91 THEN X$=CHR$(32+ASC(X$))
3430 A$=A$+X$
3440 NEXT
3450 SK=INSTR(B$,A$)
3455 RETURN
3460 :
4000 REM — Rutina para hallar atributo:
4010 AT=0
4020 IF QT=0 THEN RETURN : REM ¡No sirve!
4030 A%=0
4040 REM **** HALLAR ATRIBUTO ****
4050 AT=AT+1:WD$=""
4070 REM **** COMPROBAR CADA SINONIMO ****
4075 Y%=0
4080 A%=A%+1:X$=S$(A%)
4088 IF WD$="" THEN WD$=X$
4090 IF X$<>"*"THEN N$=X$:GOSUB 3300:Y%=SK
4100 IF X$<>"*" AND Y%=0 THEN 4070
4120 IF Y%=0 AND AT<=7 THEN 4040
4130 IF AT>7 AND Y%=0 THEN AT=0: REM fallido.
4133 IF AT=0 AND QT=3 THEN AT=1
4140 RETURN
4150 :
5000 REM — Maquina respondiendo:
5010 IF QT * AT=0 THEN RETURN
5020 IF QT=3 OR AT=3 THEN PRINT DB$(4,C0%);" es el presidente de ";
     DB$(1,C0%);"."
5030 IF QT=3 AND AT=1 OR AT=3 THEN RETURN
5050 PRINT "El ";WD$;" de";DB$(2,C0%);" es ";DB$(AT+1,C0%);
5060 IF AT>3 AND AT<7 THEN PRINT " millones de libras";
5070 PRINT "."
5080 RETURN
5200 :
8000 :
8010 REM informacion sobre las empresas:
8020 DATA BP, British Petroleum Co., Petroleo, PI Walters,
     34583,17306,5589,145150, Reino Unido
8030 DATA Shell UK,Shell Transport & Trading, Petroleo,Sir Peter
     Baxendall,21910,11962,3246,111111,Reino Unido
8040 DATA BAT,"B.A.T. Industries", Tabaco,"P.
     Sheehy",11318,4607,1018,178000,Reino Unido
8050 DATA ICI,Imperial Chemical Industries, Petroquimicos,"J.M. Harvey-
     Jones",7358,5421,724,123800,Reino Unido
8060 DATA Shell,Royal Dutch Shell,Petroleo,JM
     Raisman,6665,3704,1206,19027,Holanda
8070 DATA Esso,Esso Petroleum Co., Petroleo,AW Forster,6109,2891,1315,7628,
     USA
8080 DATA Unilever,Unilever plc,Alimentacion,K Durham,5447,2434,406,69233,Reino
     Unido
8090 DATA Imperial,Imperial Group,Tabaco,GC Kent,4614,1124,192,101300,Reino
     Unido
8100 DATA P & O, P & O Steam Navigation Co.,Naviera,JM
     Sterling,4206,927,77,12512,Reino Unido
8110 DATA GEC,General Electric Co.,Ingenieria electrica,Arnold
     Weinstock,4190,2133,621,188802,Reino Unido
8120 DATA Grand Met,Grand Metropolitan, Hoteles,SG
     Grinstead,3849,2350,366,129454,Reino Unido
8130 DATA RTZ,Rio Tinto Zinc Corporation,Mineria,Sir Anthony
     Tuke,3680,5163,492,70314,Reino Unido
8140 DATA Ford,Ford Motor Co., Vehiculos automotores,SEG Toy,
     3287,2143,278,69500,USA
8150 DATA British Leyland,British Leyland plc,Vehiculos automotores,Sir Michael
     Edwardes,3072,1346,-102,105062,
     Reino Unido
8160 DATA GWH,George Weston Holdings,Alimentacion, GH
     Weston,2981,817,157,72832,Reino Unido
8170 DATA Berisford,S & W Berisford,Mercancias,ES
     Margulies,2729,788,87,5190,Reino Unido
8440 DATA "*"
8500 REM **** LEER DATOS SINONIMOS ****
8510 DIM S$(62)
8520 FOR P%=1 TO 61:READ S$(P%):NEXT P%
8530 RETURN
8540 :
9000 REM — sinonimos:
9010 DATA,empresa,firma,organizacion,corporacion,*
9020 DATA negocio,actividad,comercio,industria,indole,
     manufactura,*
9030 DATA presidente,jefe,supremo,director,encargado,cabeza,
     lider,lleva,quien,*
9040 DATA movimiento total,cantidad,ventas,bruto,ingresos,
     cuantia,*
9050 DATA capital,dinero,accion,valor,interes,efectivo
9060 DATA beneficio,impuesto,hecho,perdidas,perder,realizar,
     devolver,ganar,cuanto,*
9070 DATA mano de obra,empleados,emplear,trabajo,personas,
     cuantos,obrero,*
9080 DATA pais,nacion,reino unido,britanica,extranjera,control,
     origen,donde,*
9090 :
```

Complementos al BASIC

El programa está escrito para el BBC Micro. Para el Commodore 64 y el Spectrum, introducir las siguientes modificaciones:

Commodore 64:

```
3450 SK=0:L=LEN(A$)
3451 FOR T=1 TO L-1
3452 FOR R=1 TO L
3453 IF MID$(A$,T,R)=B$ THEN SK=T:T=L:R=L
3454 NEXT R:NEXT T
```

Spectrum:

Eliminar todos los signos % de los nombres de las variables. Reemplazar LAST% por LA, DB$(,) por D$(,) y WD$ por W$. Introducir estos cambios:

```
1020 DIM D$(9,32,30)
3360 LET X$=Q$(P TO P)
3370 IF CODE(X$)>64 AND CODE(X$)<91 THEN LET
     X$=CHR$(CODE(X$)+32)
3410 LET X$=N$(P TO P)
3420 IF CODE(X$)>64 AND CODE(X$)<91 THEN LET
     X$=CHR$(CODE(X$)+32)
3450 LET SK=0:LET L=LEN(A$)
3451 FOR T=1 TO L
3452 FOR R=1 TO L
3453 IF A$(T TO T+R)=B$ THEN LET SK=T:LET T=L:LET R=L
3454 NEXT R:NEXT T
4090 IF X$(TO 1)<>"*" THEN N$=X$:GOSUB 3300:LET
     Y=SK
4100 IF X$(TO 1)<>"*" AND YY=0 THEN GO TO 4070
8510 DIM S$(62,20)
```

Maestro

Antes de "conocer" la música, debe saber que el juego que presentamos emplea funciones propias de microordenadores concebidos según el estándar MSX

Este programa no es, en realidad, un juego; constituye, más bien, una buena demostración de las posibilidades musicales del Spectravideo. En esta creación en *la* menor de J.-S. Bach se utilizan sólo dos vías. De usted depende agregar una tercera.

```
10 REM ********************************************
20 REM *                MAESTRO                     *
30 REM ********************************************
40 KEY OFF
50 CLEAR 5000
60 SCREEN 1,2
70 A$="V5T32L32O4R32EAO5CO4BEBO5DL16CEO4G#O5E"
80 A1$="V5T32L3202A1603A8G+16AEAO4CO3BEBO4D"
90 A$=A$+"O4L32AEAO5CO4BEBO5DL16CO4ABR16"
100 A1$=A1$+"L16CO3AG+EL32AEAO4CO3BEBO4D"
110 A$=A$+"O5L32R32ECEO4AO5CO4EGL16FAO5DFL32FD"
120 A1$=A1$+"L16CO3AO4CO3AL32O4DO3AFADFO2AO3CO2
    L16B"
130 B$="O4BO5DO4GBDFL16EGO5CE"
140 B1$="O3DGBL32BGEGCEO2GBL16AO3CL32DFO2BO3DO2
    L16GBO3L32CEO2AO3CO2L16FDL32GO3GFGCGO4CEDO3G
    O4DF"
150 B$=B$+"L32ECO4AO5CO4L16FO5DL32DO4BGBL16EO5C
    L32CO4AFAL16DBO5C8R8"
160 B$=B$+"O4L32R32GO5CEDO4GO5DFL16EGO4BO5GL32CO4GO5CEDO4G"
170 B1$=B1$+"L16ECO3BGL32O4CO3GO4CEDO3GO4DFL16EC8"
180 C$="O5DFL16ECGE"
190 C1$="R16R32L32GEGCEO3GB"
200 C$=C$+"O6L32CO5AEACEO4AO5CL16DF+AO6C"
210 C1$=C1$+"L16AD4CEGL32F+ADF+O3AO4DO3F+A"
220 C$=C$+"L32O5BGDGO4BO5DO4GBO5L16CEGBL32AF+"
230 C1$=C1$+"L16GBO4DF+L32EGCEO3GO4CO3EGL16F+"
240 D$="D+F+O4BO5D+O4F+AL16GO5GL32GECE"
250 D1$="ABO4D+L32R32ECEO3AO4CEG"
260 D$=D$+"O4L16AO5F+L32F+DO4BO5DO4L16GOSEL32ECO
    4AO5C"
270 D1$=D1$+"F+DO3BO4DO3GBO4DF+ECO3AO4CO3F+AO4C16"
280 D$=D$+"O4F+O5GF+ED+F+O4BO5D+E8R8L32"
290 D1$=D1$+"CO3BO4CO3AL16BO2BO3L32EO4EO3BGEO2B
    GB"
300 E$="R32GB-GEGC+EGEC+EO4AGFEDO5FA-FDFO4BO5DF
    DO4BO5DO4GFEDCO5EGECEO4AO5CD+CO4AD5CO4F+ED
    +C+O3BO5D"
310 E1$="L16EO3EGB-C+8R8O4DO3DFA-O2B8R8O4CO3CEF+
    O2A8R8O3B"
320 F$="FDO4BO5DO4G+BO5DO4BG+BER32R16R32EAO5CO4B
    EBO5DL16CO4AG+EL32AO5CECO4AO5CO4F+AO5CO4AF
    +AD+O5CO4BAG+B"
330 F1$="O2BO3DFO2G+8R3204L32DCO3BO4L16CO3AG+EL
    32AEAO4CO3BEBO4DCEAECEO3AO4CO3F+AO4CO3AF+AD+F
    +L16E"
340 G$="O5DO4BG+BDFG+FDFO3BO4FEDCEAECEO3AO4CD+
    CO3AO4CO3F+O4CO3BAL16G+O4BG+ER32L32EAO5CO4BE
    BO5DCO4A"
350 G1$="G+BG+EO2BG+EAO3CECO2AO3CO2D+8R32L3203
    BG+EDBG+DL16CEO2G+O3EO2A"
360 H$="O5CEDO4BO5DFECEGFEDCO4BO5CDEFDG+DBDCAF
    DO4BO5DO4G+BO5CO4AEABG+AECEO3AB"
370 H1$="O3F+O2BO3G+CADB-G+FDO2BG+ADEFD+EO3E
    O2A8A8"
380 FOR I=1 TO 10
390 XS=X$+CHR$(0)
400 NEXT I
410 X$=X$+CHR$(30)+CHR$(63)+CHR$(127)+CHR$(127)+
    CHR$(63)+CHR$(30)
420 XS=X$+CHR$(192)+CHR$(160)+
    CHR$(144)+CHR$(144)+CHR$(144)+CHR$(160)
430 FOR I=1 TO 8
440 X$=X$+CHR$(128)
450 NEXT I
460 X$=X$+CHR$(0)+CHR$(0)
470 SPRITE$(1)=X$
480 GOSUB 660
490 PLAY A$,A1$
500 GOSUB 660
510 PLAY B$,B1$
520 GOSUB 660
530 PLAY C$,C1$
540 GOSUB 660
550 PLAY D$,D1$
560 GOSUB 660
570 PLAY E$,E1$
580 GOSUB 660
590 PLAY F$,F1$
600 GOSUB 660
610 PLAY G$,G1$
620 GOSUB 660
630 PLAY H$,H1$
640 GOSUB 660
650 GOTO 490
660 FOR I=1 TO 15
670 X=RND(1)*240
680 Y=RND(1)*176
690 PUT SPRITE I,(X,Y),RND(1)*15,1
700 FOR J=1 TO 80
710 NEXT J
720 NEXT I
730 RETURN
```

Reglas del juego

Al iniciar esta serie dedicada a desarrollar un programa para jugar al "go", proporcionamos las reglas de este milenario juego japonés

Los buenos programas para jugar al ajedrez ya son bastante comunes y los investigadores dedicados al estudio de la inteligencia artificial han volcado su interés en juegos más complejos. Uno de éstos es el *go*. Si bien las reglas de este juego oriental son muchísimo más sencillas que las del ajedrez, se lo suele considerar como muchísimo más sofisticado. Tras más de 20 años de investigaciones, los mejores programas aún juegan a un nivel apenas superior al de los recién iniciados.

A lo largo de la serie desarrollaremos un programa para jugar al *go*. Aún no siendo sumamente eficaz, el programa proporciona una buena introducción al juego y se erigirá en un valioso oponente para el recién iniciado. El programa se ha diseñado específicamente para que sea fácil modificarlo y desarrollarlo.

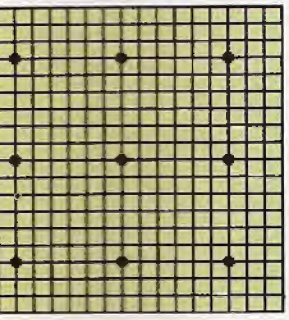
El *go* es un juego para dos personas que se desarrolla en un tablero de 361 intersecciones (fig. 1). Los dos jugadores, que juegan el uno con fichas blancas y el otro con negras, colocan alternativamente una de ellas en cualquier intersección vacante del tablero. Observe que las fichas se colocan en las uniones de líneas y no en los cuadrados.

El objetivo del juego es rodear territorio (intersecciones vacantes) con las propias fichas de cada jugador. Para ganar, las fichas de uno deben hallarse rodeando el territorio más grande al final del juego.

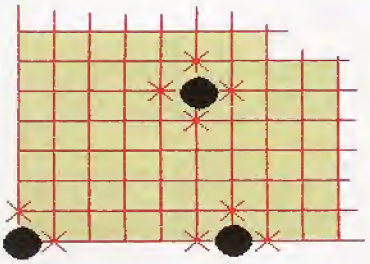
En *go* el primer movimiento siempre lo realizan las negras, que por lo general corresponden al jugador más débil. De haber una diferencia significativa entre las destrezas de juego de los dos jugadores, el primer movimiento de las negras consistirá en colocar entre dos y nueve fichas de *handicap* en el tablero, situándolas de acuerdo a un patrón específico en los nueve puntos marcados en el tablero (fig. 1).

Limitarse a rodear territorio es excesivamente fácil; también se pueden capturar fichas y eliminarlas del tablero. Todo punto vacante inmediatamente adyacente a una ficha (a lo largo de una línea) se denomina *licencia*. Por consiguiente, cada ficha individual puede tener dos, tres o cuatro licencias, según la posición que ocupe en el tablero (fig. 2). Para poder capturar una ficha del oponente, usted debe colocar su propia pieza en todas las licencias de la ficha del oponente. En la figura 3 vemos que, si el último movimiento de las negras fue ocupar la última licencia de la ficha blanca, esta última sería eliminada del tablero. Todas las fichas capturadas se suman al marcador del jugador.

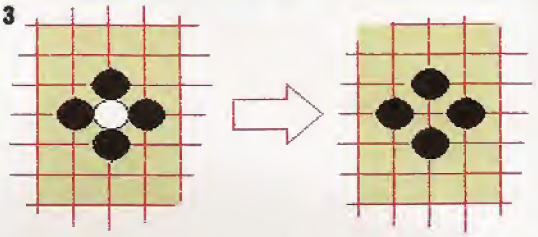
1



2

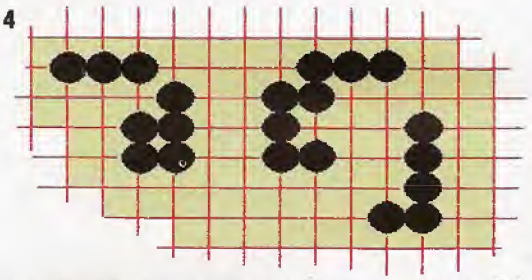


3



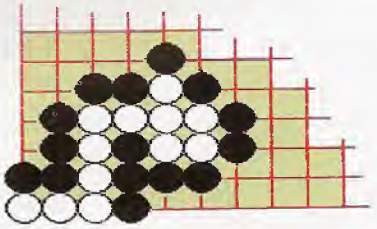
Si uno o más puntos adyacentes de una ficha se ocupan con otras fichas del mismo color, se dice que las mismas están conectadas y que forman un *grupo*. Las conexiones en diagonal no forman enlaces entre fichas: en la figura siguiente, por ejemplo, vemos cuatro grupos, y no tres:

4



Capturar grupos es más difícil que capturar fichas individuales, porque se han de eliminar como si se trataran de una sola unidad. Por tanto, las negras pueden tener que ocupar 15 puntos de licencia para poder capturar un grupo de fichas blancas.

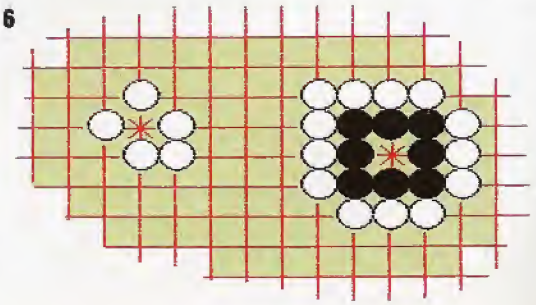
5



Esto en realidad es todo lo que se necesita saber para poder jugar al *go*, pero existen algunas complicaciones derivadas de las reglas reseñadas:

Suicidio: A tenor de las reglas, es posible que un jugador coloque una ficha de modo que no tenga ninguna licencia. Por ejemplo, si les tocara jugar a

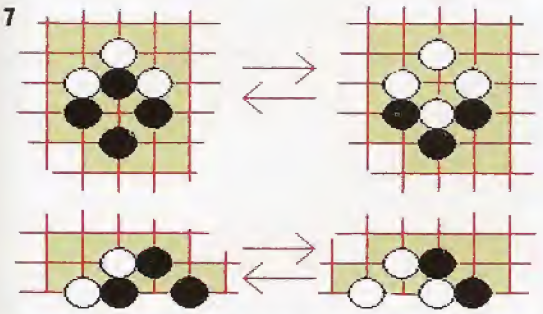
6



las negras, cualquiera de los puntos marcados en la figura 6 produciría esta situación. Este movimiento, denominado *suicidio*, está prohibido en el *go*.

Ko: Significa "infinitud" y alude a una posible situación en la que las fichas se podrían colocar y capturar indefinidamente. Los dos diagramas siguientes son casos comunes de esta situación.

7



Para impedir esta situación, las reglas del *go* prohíben que un jugador coloque una ficha de modo tal que se produzca la posición "inmediatamente anterior". Esta regla hace efectivas lo que se conoce como *luchas Ko*. Si no se permite que las blancas recapturen inmediatamente una ficha debido a la regla Ko, entonces, obviamente, se debe jugar una ficha intermedia. Ésta generalmente se coloca en una posición que amenace ya sea a una ficha negra o bien a un grupo, en algún otro lugar del tablero, y se conoce como un *Sente* (forzar una respuesta). Por tanto, las negras no pueden ocupar el punto Ko, que da por terminada la lucha Ko, y las blancas pueden recapturar la ficha Ko negra. Ahora las negras tienen prohibida la recaptura inmediata y deben tratar de hallar un movimiento Sente si es que quieren que la lucha Ko continúe.

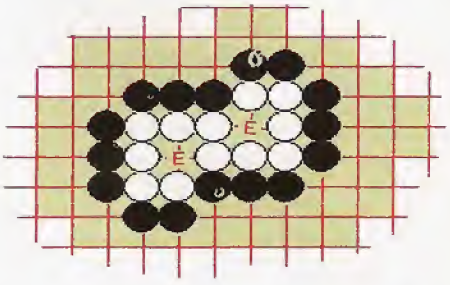
Un concepto muy importante que se desprende de estas reglas es el de *"vida y muerte"* para un grupo de fichas. Ya hemos visto que es imposible que las negras jueguen en la intersección vacante en el medio del grupo (fig. 6). Observe, sin embargo, que si aquí hubieran de jugar las blancas, el grupo negro sería capturado.

También sucedería que si el grupo negro no estuviera rodeado por fichas blancas, luego éstas no podrían colocar una ficha en este punto. Se desprende que si las blancas quisieran capturar este grupo negro, la última ficha blanca colocada debería hallarse en este punto vacante tras haber rodeado por completo el grupo negro con fichas blancas. El grupo negro sería, entonces, eliminado del tablero, dándole a la ficha blanca recién colocada cuatro puntos de licencia.

Se dice que el punto que se halla rodeado por fichas del mismo color de esta manera es un *ojo* y sucede que el ojo debe ser siempre la última licencia ocupada para capturar el grupo.

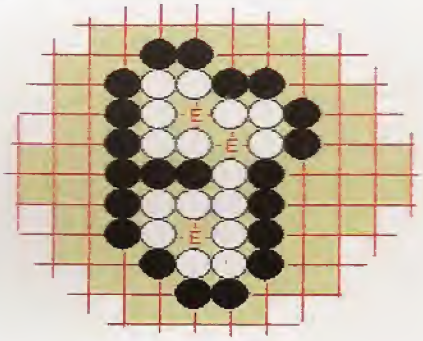
Avanzando un paso más este concepto, podemos imaginar un grupo que contenga dos ojos, como el grupo blanco que aparece en la figura 8. Para capturar al grupo, la última ficha negra debe ocupar ambos ojos simultáneamente; pero esto es imposible, ¡porque las negras sólo pueden colocar una ficha por vez! En consecuencia, este grupo (y todo el que contenga al menos dos ojos) se halla a salvo de captura, y permanecerá en el tablero hasta el final del juego (¡siempre y cuando, por supuesto, las blancas no incurran en el error de ocupar los ojos!).

8

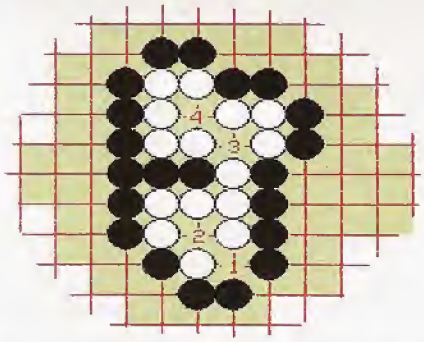


Ni siquiera es necesario que los ojos se hallen en un mismo grupo. Los tres grupos blancos de la figura 9 comparten tres ojos, y todas las fichas blancas están a salvo de ser capturadas. No obstante, usted debe tener cuidado. Mediante la secuencia indicada, las negras pueden capturar la formación similar de la figura 10 (que se diferencia en sólo una ficha). De los tres ojos, el de más abajo en realidad puede ser infiltrado por las negras; esto es lo que se conoce como *falso ojo*.

9



10



Todos estos grupos han tenido formaciones de ojos totalmente desarrollados, pero en la práctica no es necesario formar dos ojos para cada grupo, sino meramente contar con el potencial para hacerlo ante un eventual ataque.

El juego termina con el mutuo consentimiento de los jugadores, cuando ambos consideran que ninguna de las partes puede ganar ya nada más. Este final más bien ambiguo ha sido causante de muchos problemas en los programas de *go*, a los que aún les resulta muy difícil decidir cuándo debe terminar el juego. Éste también se da por terminado si ambos jugadores "pasan" en sus movimientos, devolviéndole el control al oponente. Durante el juego se les permite a ambos jugadores pasar en cualquier momento, pero ello se hace sólo muy raramente.

¿Quién gana?
Al final del juego, se calcula así el marcador de cada jugador:

1. Se ocupan todos los puntos neutrales, denominados *damas*. Éstos los puede ocupar cualquiera de los jugadores, ya que no contarán para el marcador final.

2. Todas las fichas o grupos que no puedan evitar la captura se eliminan del tablero, como si hubieran sido capturados. Ello implica que no es necesario capturar estas fichas "muertas" durante el juego, si bien usted puede hacerlo si así lo desea. Se podría efectuar la secuencia de captura, pero no beneficiaría a ninguno de los jugadores. (Cada ficha defensiva colocada incrementaría el marcador del jugador atacante en un punto, pero este jugador también tendría que colocar una ficha, ocupando por tanto su territorio, y reduciendo su marcador en un punto.)

3. Luego se calcula el marcador de cada jugador como el número de intersecciones vacantes (el territorio) controlado por él, menos el número de fichas capturadas por el oponente. Gana quien obtiene más puntos.

Líneas melódicas

Nos hallamos en la segunda fase del proyecto: la conexión de las patillas del chip a las líneas de datos y de control del ordenador

El chip adaptador de interface para comunicaciones asíncronas (ACIA) constituye la base de la interface MIDI y es compatible con los procesadores 6502 y 6510 que utilizan el BBC Micro y el Commodore 64, respectivamente. Por lo tanto, podemos enlazar las líneas de datos de dirección y de control del procesador con las patillas del chip.

La placa final está diseñada para enchufarse directamente en la puerta para ampliación del Commodore 64. La versión para el BBC Micro debe conectarse en la puerta de tubo de la cara inferior del ordenador mediante un cable plano de 40 vías y conectores adecuados. Explicaremos por separado las conexiones que se deben realizar para cada micro. Una vez construida la placa, se pueden llevar a cabo algunas pruebas simples para asegurar que funcione correctamente. Estas pruebas implican colocar datos en los registros del chip ACIA y efectuar un simple bucle de comprobación.

Es necesaria la decodificación de direcciones para acceder a cualquier periférico que esté conectado al bus de datos principal, pero que no posea la misma cantidad de líneas de dirección que la CPU. Tales dispositivos suelen poseer una patilla *chip select* que permite que el dispositivo utilice el bus de datos cuando la línea de selección de chip está activa. Las líneas de conexión conectadas al dispositivo permitirán la selección de registros diferentes.

Las señales de selección de chip para los diversos dispositivos conectados al sistema se generan decodificando las líneas de dirección más significativas. Se puede utilizar cada combinación de bits de estas líneas de dirección para seleccionar exclusivamente un dispositivo. De este modo, las registros internos del dispositivo aparecerán en el mapa de memoria de la CPU, permitiendo acceder a ellos como si fueran posiciones normales de memoria. Sin embargo, se debe tener cuidado en asegurar que las direcciones utilizadas por el dispositivo no correspondan a registros relevantes empleados por el OS del ordenador anfitrión.

La puerta para ampliación del Commodore 64 tiene dos salidas, etiquetadas I/O1 e I/O2. Estas líneas se ponen *low* (bajo) cuando se accede a las páginas $DE y $DF, respectivamente. Conectando simplemente la línea CS2 del chip ACIA con I/O1, podemos asociar el chip a la página $DE. Debido a que el ACIA no está conectado a las líneas de dirección de la A1 a la A7, se puede acceder a los registros internos del chip mediante cualquier dirección comprendida entre $DE00 y $DEFF. Conectando A0 directamente a la patilla de selección de registro del ACIA, todas las direcciones pares accederán a los registros de datos de transmisión/recepción. La elección más obvia para las direcciones es $DE00 (decimal 56832) y $DE01 (decimal 56833).

El BBC Micro Modelo B ofrece dos posibles puertas que se pueden utilizar para conectar la interface, cada una de las cuales presenta sus ventajas y sus inconvenientes. La puerta de tubo nos permite el acceso directo al bus de datos de 2 MHz. Se proporciona una línea de decodificación de dirección, NTUBE, para las direcciones desde &FEE0 hasta &FEFF. El inconveniente es que el BBC Micro comprueba si hay presente algún dispositivo en el tubo mediante la lectura de ciertas direcciones del tubo cuando éste se enciende. Si hubiera instalado algún otro dispositivo distinto del segundo procesador, el ordenador parecerá colgado mientras espera datos provenientes del segundo procesador.

Una respuesta profesional a este problema sería conectar una de las líneas de dirección de mayor orden al CS0 en el chip. Una alternativa es conectar NTUBE a CS2 y enchufar la placa en la puerta de tubo con el ordenador encendido. Por razones de simplicidad, ésta es la solución que hemos elegido para este proyecto. Sin embargo, otra alternativa es usar el bus de 1 MHz. Las patillas marcadas NPGFC y NPGFD se proporcionan para asociar dispositivos externos en las páginas &FC y &FD y se podrían utilizar de la misma forma en que se utilizan I/O1 y I/O2 en la versión para el Commodore 64.

La utilización del bus de 1 MHz tiene dos desventajas: primero, debido a que el reloj del sistema trabaja más despacio, a 1 MHz, en las dos líneas de decodificación se producen "desperfectos". Para evitar este problema debe utilizarse un circuito de limpieza (como el que se describe en la sección 28.5 de la guía para el usuario avanzado del BBC Micro). El segundo problema es que no hay ninguna alimentación de potencia de 5 V al bus de 1 MHz. En consecuencia, la potencia debe derivarse ya sea desde la puerta para el usuario o bien desde el conector auxiliar de potencia.

Un poco de sabiduría

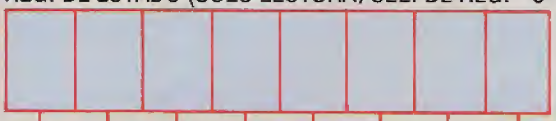
Si desea programar la interface MIDI por sí mismo, es importante que comprenda las funciones de los cuatro registros del chip ACIA. Éste se puede programar a través del registro de control para interrumpir a la CPU cuando se establezcan ciertos bits de estado en las secciones de recepción y/o transmisión del ACIA. Se producirá una interrupción del transmisor si los bits 5 y 6 del registro de control están a 1 y a 0, respectivamente, y el bit de estado TDR está establecido. La interrupción se elimina escribiendo datos en el TDR. Se produce una interrupción del receptor si tanto el bit de control 7 como el bit de estado 0 están a 1. La lectura del RDR eliminará la interrupción (a menos que también esté a 1 el bit de estado 5, que se elimina leyendo el registro de estado antes que el RDR). El bit de estado 2 también puede generar una interrupción, pero ésta

Los registros del ACIA

REG. DE DATOS TRANSMISIÓN/RECEPCIÓN:
SELECCIÓN DE REGISTRO=1

RECEPCIÓN: LEER REGISTRO
TRANSMISIÓN: ESCRIBIR REGISTRO

REG. DE ESTADO (SÓLO LECTURA) SEL. DE REG.=0



- REGISTRO DE DATOS A RECIBIR LLENO
- REGISTRO DE DATOS A TRANSMITIR VACÍO
- DETECTAR PORTADORA DATOS
- BORRAR PARA ENVIAR
- ERROR DE VENTANA (BITS INICIO–FINAL)
- RECEPTOR ATRAPADO
- ERROR DE PARIDAD (*)
- SOLICITUD DE INTERRUPCIÓN

REG. DE CONTROL (SÓLO ESCRITURA) SEL. DE REG.=0

SELECCIONAR DIVISIÓN DE CONTADOR
00 DIVIDIR POR 1 (*)
01 DIVIDIR POR 16 (*)
10 DIVIDIR POR 64
11 REINICIALIZACIÓN MAESTRA

SELECCIÓN DE PALABRA

	BITS DE DATOS	BITS DE FINAL	PARIDAD
000	7	2	PAR (*)
001	7	2	IMPAR (*)
010	7	1	PAR (*)
011	7	1	IMPAR (*)
100	8	2	NINGUNA (*)
101	8	1	NINGUNA (*)
110	8	1	PAR (*)
111	8	1	IMPAR (*)

CONTROL DE INTERR. DE TRANSMISIÓN
00 INHABILITADA (*)
01 HABILITADA
10 INHABILITADA (*)
11 INHABILITADA (*)

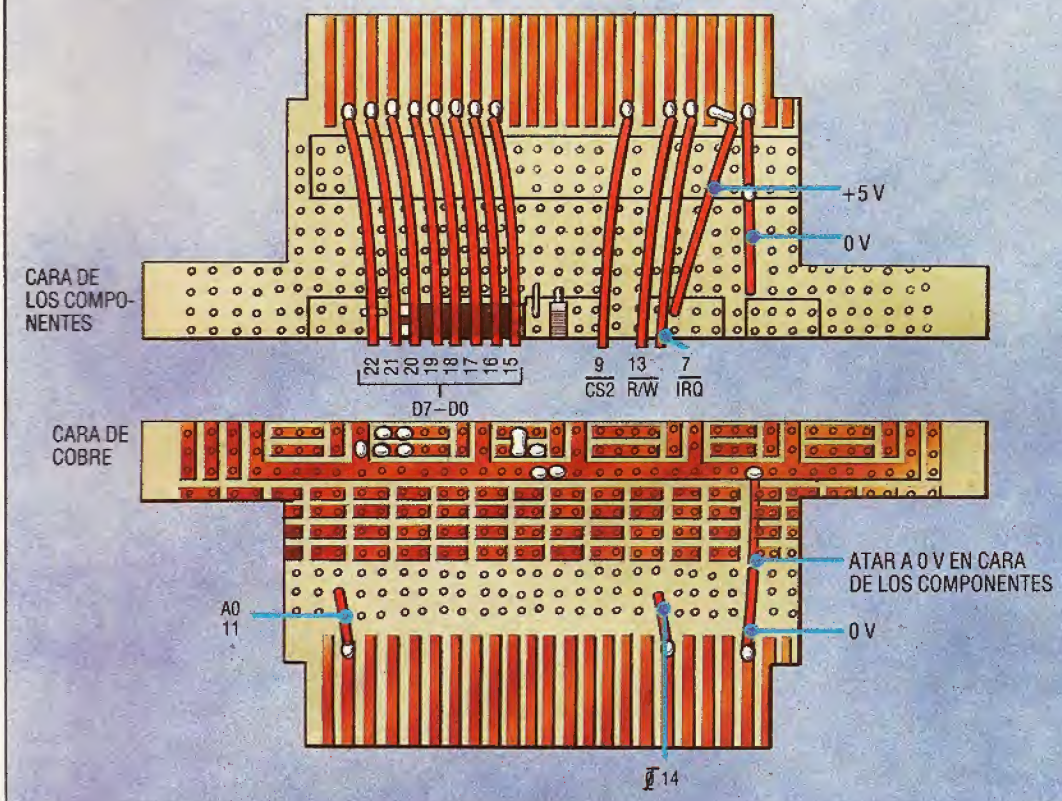
CONTROL DE INTERR. DE RECEPCIÓN
0 INHABILITADA
1 HABILITADA

(*)=NO APLICABLE AL PROYECTO MIDI

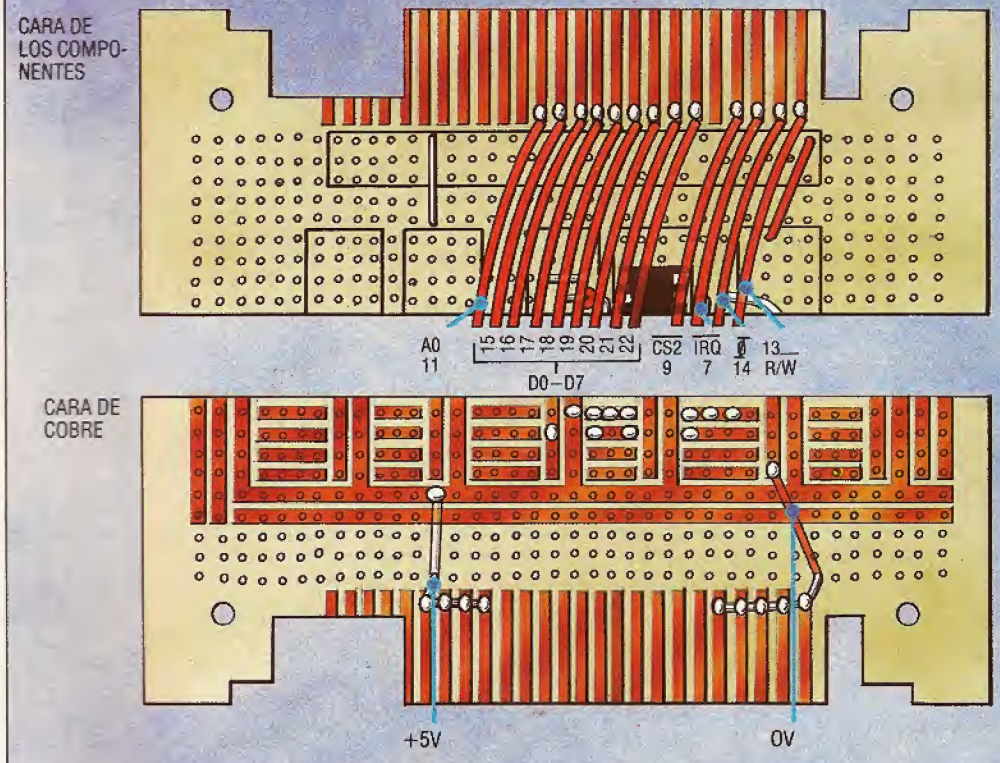
Kevin Jones

no es aplicable aquí. Se proporcionan líneas separadas para los relojes de recepción y transmisión, pero suelen estar conectadas juntas. El reloj de control de velocidad en baudios deriva directamente de la entrada del reloj o bien dividiendo por 16 o 64, según el estado de los bits de control 0 y 1. Los bits 2, 3 y 4 del reg. de control determinan la cantidad de bits de final y de bits de datos y si la paridad es par, impar o bien no se usa. Para la MIDI necesitamos los bits 2 y 4 en 1 y al bit 3 a 0 (ocho bits de datos, ninguna paridad, un bit de final) para adecuarnos al estándar. Los bits 5 y 6 del reg. de control gestionan el control del transmisor. Para la MIDI, necesitamos que el bit 6 esté a 0 y entonces el bit 5 activará las interrupciones del transmisor

Commodore 64



BBC Micro



Conexiones marginales

Estos diagramas muestran las conexiones apropiadas a las patillas del chip ACIA para cada micro. Los cables desde los conectores marginales están numerados. Éstos están relacionados con los números de patillas del chip ACIA. Observe que las patillas de este chip están numeradas del siguiente modo: sosteniendo el chip con la muesca situada arriba de todo y las patillas alejadas de usted, la patilla número 1 es la más próxima a la muesca del lado izquierdo. Las patillas se numeran luego hacia abajo por el lado izquierdo, hasta el número 12. La patilla 13 es la opuesta a la patilla 12 sobre el lado derecho, y las restantes patillas se numeran yendo hacia arriba por el lado derecho hasta la patilla 24, la más próxima a la muesca. Emplee alambre de enlace para realizar las conexiones apropiadas.

Una vez terminada, debe insertarse la versión para el Commodore 64, con el lado de los componentes hacia arriba, en la puerta para ampliación del 64, y encender el ordenador, listo ya para las pruebas. Para la versión del BBC Micro, es necesario confeccionar un cable de conexión usando un metro de cable plano de 40 vías, un conector de patillas IDC de 40 vías y un conector marginal de 20 vías

Kevin Jones

Comprobación de la placa

Con la placa ya enchufada, podemos llevar a cabo algunas pruebas sencillas para verificar el correcto funcionamiento de la placa. Será útil un tester sencillo para aislar cualquier fallo, en la improbable circunstancia de que la placa fracase en alguna de las pruebas. Si no tiene un tester, necesitará llevar a cabo una concienzuda inspección visual de la placa.

1. Si el ordenador no funciona normalmente con la placa insertada, compruebe lo siguiente:

- Verifique que el voltaje entre las líneas de +5 V y 0 V sea realmente de 5 V. De no ser así, compruebe que todos los IC estén insertados correctamente. Examine la placa por si hubiera cortocircuitos entre las pistas de potencia.
- Quite la placa del ordenador y use el tester para comprobar la existencia de cortocircuitos entre cualquiera de las conexiones del bus del ordenador.

2. Cuando el ordenador parezca funcionar normalmente, conecte un cable MIDI entre los conectores MIDI IN y MIDI OUT utilizando cables conectores DIN de *hi-fi* normales de 5 patillas. Digite la siguiente instrucción (entre paréntesis, el equivalente para el BBC Micro):

```
POKE 56832,3    (?&FEE0=3)
```

Esta instrucción coloca un 3 en el registro de control ACIA, que realiza una reinicialización maestra. Ahora digite:

```
POKE 56832,22    (?&FEE0=&16)
```

Ésta coloca el valor $16 en el registro de control y configura al ACIA del siguiente modo:

- Inhabilita las interrupciones del receptor y el transmisor (porque aún no disponemos de medios para manipularlas).
- Define las palabras en serie transmitidas y recibidas como ocho bits de datos más un bit de final sin ninguna generación/comprobación de paridad.
- Define la velocidad en baudios como el reloj en las patillas 3 y 4 dividido por 64. (2 MHz/64=31.25 KHz, que es la velocidad especificada para la MIDI).

Ahora el ACIA debe estar listo para recibir y transmitir datos. Compruébelo leyendo el registro de estado mediante:

```
PRINT PEEK(56832)
(PRINT?&FEE0)
```

Usted deberá leer el valor 2. Éste indica que tanto el registro de datos a transmitir (bit 1 a 1) como el registro de datos a recibir (bit 0 a 0) están vacíos. Puesto que no se han recibido datos y las interrupciones están desactivadas, los bits de estado restantes, del 2 al 7, deben estar a 0.

3. Envíe un byte desde el registro de transmisión a través del cable hacia el registro de recepción, mediante:

```
POKE 56833,X    (?&FEE1=X)
```

donde X es cualquier número entre cero y 255. Esta instrucción coloca un valor en el registro de datos, para transmitirlo.

4. En una fracción del tiempo que lleva digitar la siguiente instrucción, se debe haber recibido el byte. Vuelva a leer el registro de estado:

```
PRINT PEEK(56832)
(PRINT?&FEE0)
```

Ahora el valor deberá ser 3. El bit 1 se habrá borrado inmediatamente después de la última instrucción (registro de transmisión completo), pero se establecerá a 1 al cabo de un corto tiempo, apenas el registro de datos a transmitir esté listo para el siguiente byte. El bit 0 estará establecido a 1, indicando que el byte se ha recibido y que se lo puede leer desde el registro de datos. Observe que hay un posible error: si el bit 0 no está a 1, probablemente haya un circuito abierto en el camino de transmisión que esté manteniendo la entrada del receptor a un nivel *high* (alto), impidiendo su detección.

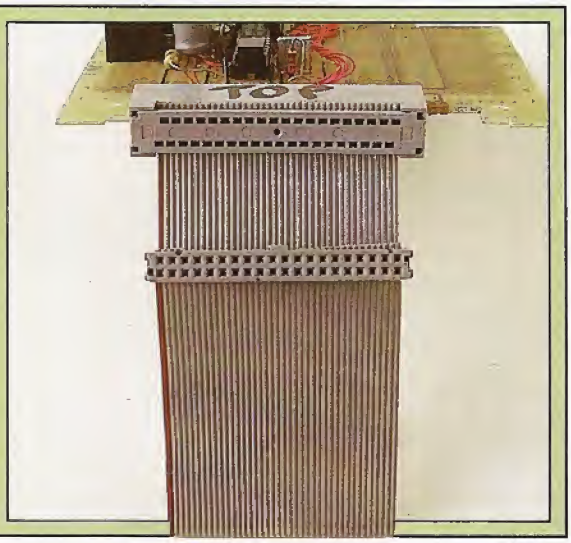
5. Habiendo verificado que se haya recibido un byte, podemos comprobar si es el mismo que el transmitido leyendo el registro de datos:

```
PRINT PEEK(56833)
(PRINT ? &FEE1)
```

Esta operación de lectura debe devolver el mismo valor X que se transmitió antes. Los pasos 3, 4 y 5 deben repetirse varias veces con distintos valores de X

Desarrollo plano

La placa de la interface MIDI se puede unir a la puerta de tubo del BBC Micro mediante un cable plano de 40 vías, un conector marginal IDC de 40 vías que se instala en el borde de la placa y un conector de patillas IDC de 40 vías. Este cable se enchufa directamente en la puerta de tubo, en la cara inferior del BBC Micro. Al montar cable y conectores, observe la orientación de los dos enchufes. Estire el cable de modo que quede plano y fije el conector marginal de modo que la patilla 1 (marcada en la carcasa del conector) quede en la fila de abajo. Haga una marca en la superficie superior de la carcasa para señalar la orientación del conector cuando sea utilizado con la placa de la interface. El conector de patillas IDC se debe fijar al otro extremo del cable con la muesca rectangular arriba (véase fotografía)



Un agente veloz

Con el Phonemark 8500 Quick Data Drive, que utiliza "wafers" de floppies serializados, los usuarios de Commodore mejoran sus condiciones de almacenamiento externo

Chris Stevens

Estimable alternativa
A pesar de las críticas de que han sido objeto el aparato de cassette y la unidad de disco de Commodore, ha habido pocos proveedores "independientes" de sistemas de almacenamiento alternativos para estas máquinas. La Quick Data Drive es un sistema de wafer flexible serializado que utiliza bucles de cinta continuos idénticos a los de la Rotronics Wafadrive. El sistema no es particularmente barato, pero vale la mitad que la unidad de disco Commodore y en muchas aplicaciones es considerablemente más rápido

Haciéndose eco de la demanda de muchos entusiastas de los ordenadores personales, que desean contar con métodos más rápidos y eficaces para acceder a sus programas, los fabricantes están produciendo una amplia gama de dispositivos de almacenamiento masivo dirigidos a los micros más populares. Entre estos dispositivos se halla el Phonemark 8500 Quick Data Drive, de Dean Electronics, fabricado para el Commodore 64 y el Vic-20.

Este dispositivo es un pariente cercano de la Rotronics Wafadrive, diseñada para el Sinclair Spectrum. Las unidades para ambos las fabricó BSR Electronics y utilizan *wafers* idénticos.

La unidad de disco estándar Commodore 1541 es conocida, al igual que la unidad de cassette, por su lentitud. Ello no se debe a la unidad de disco en sí misma sino al método que utiliza el ordenador para cargar los datos. Gran parte del sistema operativo del Commodore 64 es herencia de las máquinas de gestión PET de mediados de los setenta, cuando los sistemas de almacenamiento masivo, en particular las unidades de cassette, eran muy poco fiables como dispositivos para almacenamiento de datos. Cuando Commodore concentró su atención en proporcionar almacenamiento de apoyo para la serie PET, decidió suministrar su propia unidad de cassette e incorporar una serie de comprobaciones para asegurar que los datos cargados fueran correctos. Si bien ello incrementó la fiabilidad del proceso de carga (LOAD), lo logró a costa de la velocidad de acceso. Este sistema se traspasó al Vic-20 y posteriormente al Commodore 64.

En la actualidad, la calidad de la cinta de cassette ha mejorado mucho y la necesidad de una comprobación de datos larga y complicada para las máquinas Commodore ha desaparecido. Muchos paquetes de software comerciales contienen actualmente sus propias técnicas de carga, que eliminan muchas de las comprobaciones y hacen que la carga se realice con mayor rapidez, sin pérdida de fiabilidad.

No obstante, al cargar sus propios programas, la mayoría de los usuarios no tienen acceso a estas técnicas de gran velocidad y deben aún padecer las demoras impuestas por el sistema operativo Commodore. Se puede considerar que la Quick Data Drive, de la cual se afirma que carga 15 veces más rápido que las cassettes normales y más rápido que la unidad de disco 1541, representa una alternativa comercial a los periféricos Commodore.

La Quick Data Drive es un dispositivo bastante pequeño, de poco más de la mitad del tamaño del aparato de cassette Commodore 1530. La unidad se conecta al Vic-20 o al Commodore 64 a través del conector marginal para cassette. A diferencia de la Rotronics Wafadrive, la Quick Data Drive posee sólo una unidad. Aunque sería preferible contar con dos unidades cuando se necesita operar simultáneamente un wafer de sistema y un wafer de datos, la unidad única resulta adecuada a la mayoría de los usuarios. Si fuera absolutamente necesario, siempre se podrá adquirir una segunda unidad y conectarla para proporcionar una capacidad de unidad dual.

El hecho de que la unidad se enchufe en la puerta para cassette no significa que usted no pueda tener en funcionamiento, al mismo tiempo, un aparato de cassette. En la parte posterior de la Data Drive hay un conector marginal para cassette que permite encadenar en margarita unidades de cassette o una segunda Quick Data Drive.

Si bien la Rotronics Wafadrive y la Quick Data Drive poseen un aspecto similar, sus métodos de operación son muy diferentes y reflejan las diferencias existentes entre los ordenadores para los que fueron diseñadas. Mientras que el sistema operativo de la Wafadrive está retenido en ROM (imitando a las ROM de la Interface 1), el sistema operativo de la Quick Data Drive (QOS) está retenido en un wafer. Para cargar (LOAD) el QOS en el ordenador, se debe pulsar Shift/Run (tal como se haría para cargar una cassette normal). Cuando aparece en pantalla la indicación PRESS PLAY ON TAPE, se debe pulsar un pequeño botón situado en la parte

posterior de la unidad, que carga automáticamente el sistema. Una vez hecho esto, el QOS lo llevará a cabo de forma automática para los subsiguientes accesos.

Los programas que componen el QOS se cargan en dos zonas separadas de la memoria. Primero se almacenan las rutinas en código máquina para cargar (LOAD), guardar (SAVE) y buscar programas, entre las direcciones C000 y CFFF hacia la parte superior de la memoria (normalmente utilizada para programas en código máquina). Es este módulo del QOS el responsable de la mayor velocidad de carga de la unidad.

Aunque el QOS no implementa ninguna instrucción propia (al utilizar aquellas ya disponibles en la ROM del sistema operativo Commodore), intercepta las rutinas que se ocupan de la carga normal e inserta las suyas propias.

QUICK DATA DRIVE

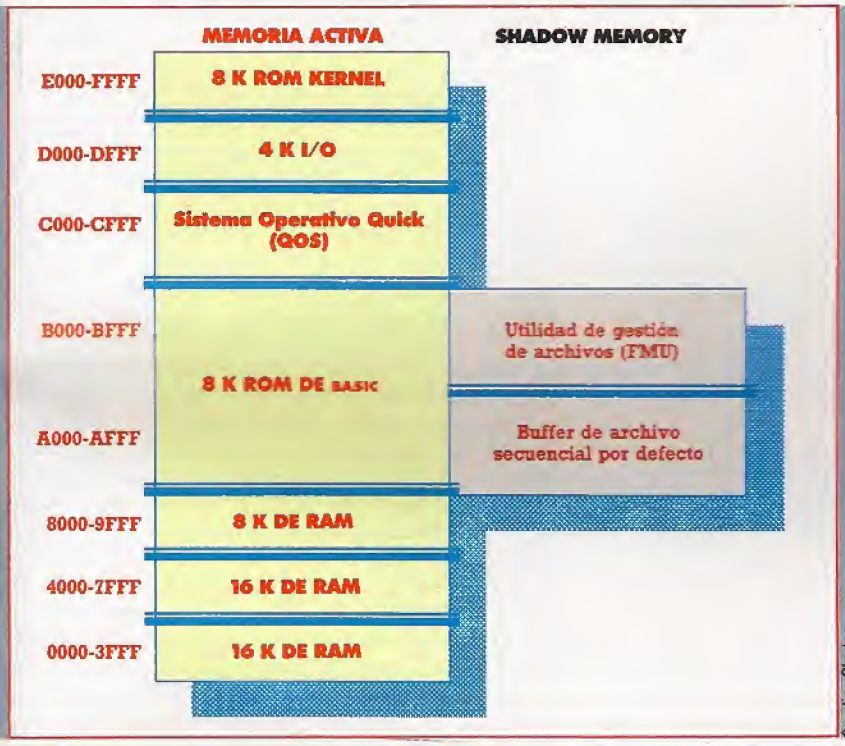
DIMENSIONES
147×118×49 mm

INTERFACES
Conector para la puerta de cassette del Commodore 64 y el Vic-20

FORMATO
Wafers flexibles serializados de bucle continuo

CAPACIDAD
A la venta wafers de 16, 64 y 128 Kbytes

VELOCIDAD
Tiempo de acceso promedio: 8 seg/archivo de 15 Kbytes; 43 seg/archivo de 120 Kbytes



Caroline Clayton

La otra parte del Quick Operating System (QOS) es la FMU (*file management utility*: utilidad para gestión de archivos), que contiene numerosas rutinas útiles. Ésta se mantiene en la mitad superior de los 8 K de la Shadow Memory por debajo de la ROM de BASIC, entre las direcciones B000 y AFFF. Los 4 K inferiores entre A000 y AFFF se utilizan como un tampón o buffer de archivos secuenciales que emplea la FMU. Debido a que la FMU está almacenada en la zona de memoria por debajo de la ROM de BASIC, no se pueden utilizar ambas al mismo tiempo. Para llamarla desde la memoria debe ejecutarse la instrucción LOAD 'FMU'. Ésta simplemente "blanquea" la ROM de BASIC y da entrada a la FMU.

Sin ninguna duda, las operaciones de la Quick Data Drive son mucho más rápidas que los métodos de cassette estándares. A modo de ejemplo, se utilizó como programa de comprobación el juego de simulación *El Nuevo Mundo*, desarrollado en nuestro apartado "Programación". Cargar el programa completo de 25 K, lo que consumía más de nueve minutos desde cassette y un minuto y medio desde disco, llevó apenas 30 segundos con la Quick Data Drive, lo que representa una mejora considerable. No obstante, coomo con todos los sistemas basados en bucle de cinta, ello depende en gran medida de dónde se halle la cabeza de lectura/escritura de la unidad en relación al comienzo del programa.

El Sistema Operativo Quick encuentra los archivos de datos requeridos comprobando cada uno de los bloques de encabezamiento de la cinta. Cuando se formatea un wafer, el sistema operativo divide la cinta en bloques, cada uno con su propia sección de nombre de archivo. Cuando se requiere cargar un archivo en el ordenador, busca hasta hallar el bloque que contenga la primera parte del archivo, lo carga y después busca el segundo.

Del mismo modo, cuando se solicita visualizar un directorio del wafer, el QOS lee cada uno de los nombres de archivo a medida que el cabezal de lectura/escritura va pasando por la cinta, y registra los bloques que contienen archivos y aquellos que están vacíos. Cuando se han leído todos los nombres de archivo, el sistema visualiza la lista de archivos más la cantidad total de espacio disponible expresada en bytes.

Gestión de archivos

La FMU, utilidad para gestión de archivos, es un conjunto de rutinas activadas por menú que cubren aplicaciones tales como formateo y lectura del directorio del wafer. Asimismo, contiene rutinas para copiar que permiten transferir datos desde disco, cassette o wafer a un wafer de seguridad. Ésta es, obviamente, una importante característica del sistema Quick Data Drive, porque pocos programadores pensarían en adquirir un sistema de almacenamiento, independientemente de lo bueno que fuera, que no pudiera transferir programas existentes al nuevo soporte.

Por supuesto, el sistema tiene sus inconvenientes. Si bien las rutinas para copiar funcionan bien para programas en BASIC y archivos secuenciales, ciertas rutinas (en especial en código máquina) que se cargan en una zona específica de la memoria suelen causar problemas. Ello se debe a que las dos zonas que utilizan el QOS y la FMU son también las zonas en que se conserva el código máquina. Por lo tanto, cuando se cargan programas en código máquina, éstos se graban sobre el sistema operativo de la Quick Data Drive y el programa queda colgado.

Esta característica de diseño significa que si bien se pueden copiar los programas del usuario, aún no es posible la transferencia de software comercial. No obstante, de venderse suficientes cantidades de dispositivos Quick Data Drive, sin duda se producirá un programa para permitir dicha transferencia.

Hasta entonces queda en manos de Dean Electronics el persuadir a las casas de software de que vendan sus juegos y aplicaciones en wafers. En este sentido, quizá la Rotronics Wafadrive resulte ser un valioso aliado: las empresas de software podrán adquirir, entonces, cantidades mayores de wafers con mayores descuentos, cuando descubran que pueden colocar tanto su software para el Commodore 64 como para el Spectrum en el mismo soporte de almacenamiento.

Una rápida memoria
La Quick Data Drive no conserva su sistema operativo en ROM sino que ha de cargarlo desde un wafer de sistema. Los diversos componentes se instalan en dos zonas separadas de la memoria. El primero, el Sistema Operativo Quick (QOS: *quick operating system*), se carga en la zona normalmente reservada a programas en código máquina. La otra parte del sistema, la utilidad de gestión de archivos (FMU: *file management utility*) se retiene en la "RAM en sombras", detrás de la ROM de BASIC. Aquí también se halla contenido un buffer de archivos en el cual se cargan los programas antes de guardarlos en otro soporte

Código que crea código

"Last One" y "Sycero" son dos generadores de programas que ejemplifican la situación actual del software que escribe software

Enchufando

Todos los programadores recién iniciados (y probablemente también los más experimentados) sueñan con la existencia de un programa que elimine de la programación el trabajo pesado y la frustración y acabe con unas líneas de código que se ejecuten perfectamente, a la primera, sin ninguna clase de depuración ni mensajes de error.

En cierto sentido, por supuesto, quien escribe un programa en BASIC ya está utilizando un programa para escribir programas: bien resida en ROM o bien haya que cargarlo desde disco o cinta, el intérprete de BASIC toma las líneas que digita el usuario y las reescribe en código máquina para que el ordenador pueda comprenderlas. Éste es un proceso mucho más sencillo que entrar los propios dígitos del código máquina, desprovistos aparentemente de cualquier significado.

La mayoría de los programadores estarán familiarizados con el truco de hacer que un programa genere líneas de programa extras ya sea utilizando el buffer del teclado con un contador creciente para números de línea, o bien colocándolas (POKE) en la memoria. Esto con frecuencia lo emplean los programadores de código máquina para atisbar (PEEK) las posiciones de memoria y escribirlas en un programa cargador en forma de sentencias DATA.

Algunos lenguajes, por ejemplo el COMAL, poseen comprobación de errores incorporada y se niegan a aceptar una línea de código que no se haya entrado de la forma correcta, de modo que usted no podrá colar un PRNT cuando lo que quiere significar en realidad sea PRINT. El FORTH rechazará una instrucción que emplee una palabra que no esté incluida en su vocabulario básico o que el usuario no haya definido previamente. El LOGO posee una capacidad similar para funcionar de acuerdo a sus propias limitaciones.

Sin embargo, las facilidades que acabamos de mencionar continúan sin permitir que el usuario, por ejemplo, le pida al ordenador que escriba un programa para calcular su impuesto sobre la renta. Pero el crecimiento de sistemas que imitan o parecen incorporar cierto nivel de inteligencia artificial aparece como una posibilidad menos remota. Por ejemplo, hay un programa llamado *Microtext*, que se puede utilizar para guiar a un usuario no técnico a través de complicadas tareas técnicas. También se lo puede emplear para generar ayudas interactivas para la enseñanza, programas que busquen fallos, cuestionarios y software para recuperación de información. Mediante el empleo de un sistema de "autor" para crear una aplicación, disponiendo durante la comprobación de un editor en pantalla activo, el software resultante se puede "publicar" y convertir en un sistema de sólo ejecución que el usuario no pueda modificar.

Por lo general el *Microtext* sólo está a disposición de las máquinas CP/M más costosas (haciendo de la disponibilidad de un sofisticado editor de pantalla una valiosa facilidad), pero también ha salido al mercado a un precio muy inferior para el Tatung Einstein.

"The Quill" e "Illustrator"

Otro generador de programas notablemente refinado es *The Quill* (La pluma), diseñado específicamente para quienes escriben juegos de aventuras. Permite que usted prepare y edite una base de datos de información (escenarios, objetos, cómo se los puede manipular, etc.) y la incorpora a un programa en lenguaje assembly que se ejecuta de buenas a primeras sin quedar colgado.

A *The Quill* recientemente se le ha unido *Illustrator*, que añade gráficos a las aventuras de texto. *The Quill* está destinado al Spectrum, el Amstrad 464, el Commodore 64 y el Oric, e *Illustrator* al Spectrum y al Amstrad, y pronto se editará también para Commodore. A pesar de su bajo costo, *The Quill* es lo suficientemente bueno como para haber sido utilizado para escribir muchos juegos de aventuras comerciales, entre los que destaca en particular *Hampstead*, de Melbourne House.

Aunque técnicamente ambos son generadores de programas, para la industria del micro existen sólo dos programas que realmente escriben programas: *The Last One* (El último), de nombre más bien ostentoso y que en el momento de su lanzamiento se anunció como el último software que usted necesitaría comprar en toda su vida, y el *Sycero*, bastante más potente aunque algo menos amable.

The Last One, o *TLO*, como prefieren llamarlo sus distribuidores, se escribió originalmente para máquinas CP/M de 64 K. Esto es evidente, ya que posee la misma clase de alusión constante a los *overlays* de disco que hacen que el *WordStar* sea tan lento. Ahora existe para máquinas MS-DOS con un mínimo de 256 K de RAM, pero si bien ello permite la adición de muchas facilidades más sofisticadas, continúa operando a un ritmo ligeramente lento. Sus autores afirman haber incorporado en el programa un cierto grado de inteligencia artificial.

La tecla Escape

Juegos de generación
Estas pantallas ilustran las clases de programas que se pueden producir con el *Microtext*. El programa de comprobación de enchufes, a la izquierda, formula al usuario preguntas sobre sistemas eléctricos. El programa de la derecha pone a prueba su conocimiento del procedimiento correcto a seguir cuando uno se encuentra atrapado en un hotel en llamas. Observe que ambos programas dependen de la interacción del ordenador con el usuario, comparando las respuestas del usuario con una base de datos de conocimientos retenida en su memoria. Esta clase de aplicaciones es ideal para los generadores de programas

The Last One: Para el Apple II y IIe, máquinas CP/M-80 y MS-DOS

Distribuido por: D J "Al" Systems Ltd, Summers Orchard, Speke Close, Ilminster, Somerset TA19 9BJ, Gran Bretaña

Microtext: Para el Tatung Einstein y máquinas CP/M-80

Distribuido por: Transdata Ltd, 11 South Street, Havant, Hants, Gran Bretaña

The Quill: Para Spectrum, Commodore 64, Amstrad 464 y Oric

Distribuido por: Gilsoft, 30 Hawthorne Road, Barry, South Glamorgan CF6 8LE, Gran Bretaña

Sycero: Para máquinas MS-DOS

Distribuido por: System C Ltd, 7 Mill Street, Maidstone, Kent ME15 6XW, Gran Bretaña

Ciertamente, si usted lo carga con un tipo de BASIC distinto del que está buscando, identificará correctamente el disco DOS en cualquier mensaje de error.

Aunque similares a nivel superficial (en tanto y en cuanto ambos son generadores de programas), *Sycero* y *The Last One* en realidad son bastante diferentes en cuanto a concepción, esfera de acción y la forma en que abordan el problema de esclarecer las necesidades del usuario con el fin de producir código utilizable (y transportable). Asimismo, son similares en el hecho de que ambos generan programas en BASIC y que son particularmente idóneos para la entrada, proceso, almacenamiento, recuperación e impresión de datos. Los dos se diferencian de otros programas que realizan sofisticadas funciones de base de datos, porque el código resultante se ejecuta independientemente del sistema generador.

En cierto sentido, *TLO* está dirigido a las necesidades de los usuarios de hoy en día, porque da por sentado (probablemente de forma correcta) que usted prefiere hacer todo el trabajo en pantalla, prescindiendo de preparativos y cálculos en hojas de papel. Pero utilizando *Sycero* y *TLO* de esta manera no se aprovecha cabalmente su potencial. El plan de acción de *Sycero*, de siete puntos, hace hincapié en la importancia de la planificación preliminar:

- Planificar
- Especificar sistema
- Dibujar pantallas
- Comprobar datos
- Definir programa
- Producir salida impresa
- Generar programa

Y tanto *Sycero* como *TLO* resaltan el punto con suficiente claridad en sus manuales: la organización y la planificación preliminares son esenciales para una buena programación. El *Sycero* afirma:

> *Corregir errores originados por un mal diseño puede llevar más tiempo que escribir el sistema en primer lugar. Siempre es tentador dedicar cinco minutos a pensar en un problema y luego, con los discos Sycero en la mano, correr hacia el ordenador y elaborar el sistema sobre la marcha. Éste no es un buen enfoque.*
>
> *...usted siempre debe comenzar elaborando una lista exhaustiva de todo lo que desea que el ordenador produzca para usted, desde el punto de vista de lo que quiere ver en la pantalla (información* on-line*) y de lo que desea en cuanto a salidas impresas ("listados"). Sólo tras haber determinado lo que usted espera del sistema, habrá de decidir qué información necesita para producir esos resultados.*

TLO es menos específico pero insiste en el mismo punto:

> *Planifique el flujo global de su programa y anticipe los errores que va a cometer el usuario final. Después prepárelo todo para hacer frente a los mismos de forma segura. De hecho, diseñe su programa tan concienzudamente como planificaría cualquier otra tarea. Por regla general, es más fácil comenzar escribiendo algunos programas cortos, unidos entre sí mediante un menú sencillo que llame a cada uno de ellos por turno cuando así sea necesario; más adelante, hallará que este enfoque modular permite planificar y crear con gran facilidad programas extensos y complicados.*

Ninguno de estos paquetes generará sólo una clase de programas. No producirán ningún tipo de juego, por supuesto (ni siquiera una aventura basada sólo en texto), ni tampoco generarán un procesador de textos ni una hoja electrónica. Sus puntos fuertes residen en permitir que usted manipule virtualmente cualquier clase de datos, efectúe cálculos con ellos (que puede ser cualquier cosa, desde una simple extracción del IVA de una suma bruta hasta una larga y compleja ecuación de ingeniería), almacenarlos, recuperarlos e imprimirlos. En este sentido, ambos son sumamente poderosos.

Es materia de controversia el hecho de que trabajen o no tan bien como lo hace un programa directo de administración de bases de datos como el *dBase II*; pero es innegable que son más fáciles de utilizar y más amables que un programa de esta clase, cuya hostilidad suele ser directamente proporcional a su potencia.

Procesar listas

Basado en listas que pueden representar tanto datos como funciones, el LISP permite al programador adaptar el lenguaje a casi cualquier aplicación. Iniciamos una serie dedicada al LISP, en la que nos referiremos al uso que hace de las listas y veremos por qué ha tenido tanta aceptación en el campo de la inteligencia artificial.

El LISP ha tenido bastante difusión en los últimos años, fundamentalmente a raíz de su inclusión en la investigación y desarrollo en el campo de la inteligencia artificial. A medida que se acrecentaba el interés por él, se hizo evidente que el LISP podía actuar como un lenguaje para fines generales con una amplia variedad de aplicaciones.

Esta convicción generó una ingente cantidad de implementaciones del LISP con muchas aplicaciones diversas, desde la escritura de sistemas operativos y compiladores hasta la escritura de juegos de aventuras. Una lamentable consecuencia de esta proliferación del lenguaje es que, como sucede con tantos otros lenguajes, existen ahora muchos dialectos de LISP. A pesar de ello, la mayoría de las implementaciones están moderadamente estandarizadas (debido a su estructura simple y directa) y por lo general traducir el LISP de una máquina a otra es bastante sencillo.

Es probable que al utilizar el LISP advierta la carencia de una cierta función o instrucción. Esto se debe principalmente a la falta de un estándar oficial, con lo que sólo quedan las funciones que los programadores consideran más importantes. Sin embargo, tal como veremos, ampliar el lenguaje es extraordinariamente fácil, mediante la adición de las instrucciones nuevas que usted requiera.

El LISP es completamente diferente de los lenguajes más comunes, como el PASCAL, el FORTRAN y el BASIC. Posee una estructura sintáctica única, muy simple y uniforme. Esta estructura lo hace ideal para la manipulación de datos y problemas de emparejamiento.

La estructura inherente de las listas se puede adaptar fácilmente a las principales estructuras de información para ordenador, permitiendo utilizar el lenguaje para búsqueda y clasificación, funciones aritméticas e incluso para practicar juegos. Además, la mayoría de los microordenadores soportan instrucciones especiales para hacer uso de sus propias facilidades de sonido y de gráficos. Éste es, ciertamente, el caso del LISP Acornsoft para el BBC Micro, que es el que utilizaremos a lo largo de esta serie. No se preocupe si posee una versión diferente del lenguaje, ya que encontrará que todo se puede transferir directamente.

La base del LISP, como cabría esperar, es la estructura de datos de *lista*: el nombre del lenguaje proviene de los términos *"list processing"* (proceso de listas). En muchos sentidos, una lista es similar a la más familiar "matriz" de la mayoría de los micros. No obstante, a diferencia de la matriz, una lista no posee ninguna longitud específica. Cualquier lista de elementos se indica encerrándola entre paréntesis, en la forma:

```
(a b c d e ...)
```

donde a, b, c, etc., son los elementos de la lista. El término técnico para estos elementos es *átomos*, y pueden ser datos numéricos, no numéricos (datos de caracteres o variables) o incluso otra lista. Observe que los elementos separados de una lista no se separan entre sí mediante comas; en cambio, se utilizan espacios.

Las operaciones se realizan sobre las listas utilizando funciones, de forma muy similar a una función de BASIC (instrucción DEF FN), que lleva a cabo una operación sobre sus argumentos para producir un resultado. Una función se escribe de la forma:

```
(func a b c d ...)
```

donde func es el nombre de la función y a, b, c, etc. son sus argumentos. Como puede ver, la función tiene un aspecto muy parecido al del primer elemento de la lista. De modo que, por ejemplo, una lista compuesta por los seis primeros números primos se escribiría:

```
(1 2 3 5 7 11)
```

Si quisiéramos sumar estos números en BASIC, utilizaríamos:

```
1+2+3+5+7+11
```

En LISP, aplicaríamos la función plus y escribiríamos:

```
(plus 1 2 3 5 7 11)
```

que nos devolvería como respuesta 29. Una de las características de la función PLUS es que puede tomar cualquier cantidad de argumentos; de modo que podríamos igualmente escribir:

```
(PLUS 1 2 3 (PLUS 5 7 11))
```

Aquí, primero se evalúa la función PLUS más interior, que devuelve la respuesta 23. Ésta se utiliza luego como el cuarto argumento del PLUS más exterior para devolver la respuesta final 29. Un punto importante a observar aquí es la facilidad con que se pueden anidar las funciones. Anidando otra función, SETQ, podemos asignar el resultado a la variable A:

```
.(SETQ A(PLUS 1 2 3 5 7 11))
```

Implementaciones del LISP

A pesar del creciente interés que existe por el LISP, en la actualidad hay pocas versiones del lenguaje al alcance del presupuesto de los usuarios de un micro. Existe el LISP Acornsoft para el BBC Micro y el Acorn Electron a un precio razonable, si bien la mayoría de los usuarios se encontrarán con que tendrán que adquirir la Guía para el Usuario del LISP, que suma otro coste adicional.

Otros micros no están tan bien provistos. Existen, no obstante, algunas versiones CP/M del lenguaje, en particular Toolworks LISP/80 e ILISP. Hay otra versión CP/M, MULISP-83, de Microsoft, que es considerablemente más cara.

Los intérpretes basados en CP/M mencionados anteriormente se ejecutarán en la mayoría de los micros personales que soporten CP/M (Memotech, Einstein, etc.), aunque los usuarios del Amstrad no podrán utilizarlos debido a la falta de espacio de memoria (el mínimo requerido es de 48 K). Probablemente la mejor relación calidad-precio la ofrezca Toolworks, con un espacio de almacenamiento de aproximadamente 3 600 celdas de listas y 11 000 caracteres para nombres de átomos en 48 K. Todos los programas se pueden adquirir en Grey Matter Ltd, 4 Prigg Meadow, Ashburton, Devon, TQ13 7DF, Gran Bretaña

donde SETQ posee dos argumentos: la variable A y una función que halla el entero 29. Por supuesto, SETQ es en sí misma una función y por tanto debe devolver un resultado (en este caso el valor 29). En consecuencia, la siguiente asignación, que es ilegal en la mayoría de las versiones de BASIC:

```
LET B=1+2+3+5+7+11
```

se podría escribir en LISP como:

```
(SETQ B(SETQ A(PLUS 1 2 3 5 7 11)))
```

Introduciendo la función TIMES, podemos asignar, por ejemplo, 2 veces el valor de A a la variable B:

```
(SETQ B(TIMES 2(SETQ A(PLUS 1 2 3 5 7 11))))
```

En primer lugar se calcularía el PLUS para devolver el valor 29, que se le asignaría a la variable A en virtud de la función SETQ más interior. Esta función, a su vez, daría como resultado 29, a utilizar como el segundo argumento de la función TIMES. El nuevo resultado, 58, se pasaría entonces a la función SETQ más exterior para colocar el valor en la variable B. Toda la expresión daría como resultado 58, que se emplearía, entonces, para otras funciones, y así sucesivamente.

Observe que los paréntesis están proliferando mucho en el último ejemplo. Ésta es una característica del LISP, y seguir el rastro de todos estos paréntesis puede ser bastante tedioso, aunque, si el programa está bien diseñado, tienden a ocuparse de sí mismos. Además, algunos sistemas de LISP ofrecen la ayuda de informar sobre la cantidad de paréntesis que posee.

Por ejemplo, cuando se utiliza el LISP Acornsoft en el BBC Micro, la cantidad de flechas que aparecen al comienzo de la indicación de línea señala la cantidad de paréntesis que es necesario cerrar para completar la expresión.

En la última expresión introdujimos la función TIMES y le dimos dos argumentos: el valor entero 2 y una expresión de lista que previamente había dado como respuesta 29. Sin embargo, TIMES, al igual que PLUS, puede tener un número variable de argumentos. Por tanto, todas las expresiones siguientes serían legales:

```
(TIMES 1 2 4 8 16)
(TIMES 1 2 4 8 (TIMES 4 4))
(TIMES 1 2 (TIMES 2 2)(TIMES 2 4)(PLUS 8 8))
```

y todas devolverían el resultado 1024.

En realidad, la mayoría de las implementaciones del LISP poseen un límite en cuanto a la cantidad de argumentos que pueden tener este tipo de funciones. Por ejemplo, el LISP Acornsoft tiene un límite de 28 argumentos, y algunas otras versiones tienen restricciones aún más severas.

Llegados a este punto, podemos ver que una instrucción en LISP es simplemente una lista de elementos, en la cual el primero es la función a realizar, y los subsiguientes son los argumentos de la función. Éstos, a su vez, pueden ser listas con el primer elemento de cada una siendo una función que devuelva un resultado. Ahora se plantea la cuestión de qué hacer si no deseamos ninguna función. Es posible que querramos meramente preparar una lista de elementos de datos, supongamos, para utilizarla como un título.

No podemos escribir:

```
(MI COMPUTER EL MEJOR)
```

como una lista de cuatro elementos de datos, porque el LISP intentaría evaluar MI como el nombre de una función, con tres argumentos, y pensaría que cada uno de los tres argumentos (COMPUTER, EL y MEJOR) son tres nombres de variables.

Podemos decir al LISP que no evalúe una expresión anteponiéndole un apóstrofo ('), de modo que escribiríamos lo anterior como:

```
'(MI COMPUTER EL MEJOR)
```

Entonces podríamos asignarle esta lista a una variable:

```
(SETQ REV '(MI COMPUTER EL MEJOR))
```

que le asignaría a la variable REV una lista de cuatro elementos no numéricos.

Observe que en LISP no hay ningún tipo de variable.

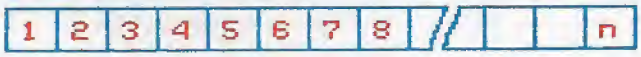
Si estuviéramos utilizando BASIC, una variable de enteros asumiría la forma A%, una variable de reales A y una variable en serie A$. En LISP las variables no se diferencian de esta manera, de modo que todas las expresiones siguientes son legales:

```
(SETQ A 3)              A=el entero 3.
(SETQ A 'COMPUTER)      A=la serie 'COMPUTER'.
(SETQ A (PLUS 2 4 8))   A=el resultado de 2+4+8.
(SETQ A '(1 2 4 8))     A=la lista (1 2 4 8).
(SETQ A '(MI COMPUTER)) A=la lista (MI COMPUTER).
(SETQ A B)              A=el valor de la variable B.
```

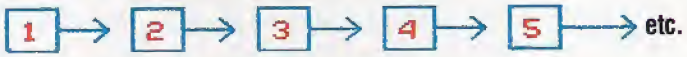
Si bien los tipos de enteros y en serie son relativamente estándares, pocos microordenadores soportan una versión de LISP que efectúe aritmética de punto flotante. En la mayoría de los casos, la aritmética de enteros es suficiente y en el improbable caso de que fuera necesaria la aritmética de punto flotante, sin ninguna duda se podría simular empleando los enteros estándares.

En este punto puede parecer difícil comprender en qué medida puede ser de utilidad el lenguaje. En el próximo capítulo veremos las funciones del LISP que pueden manipular sus argumentos de datos, y veremos de qué forma se emplean los principios de la recursión.

MATRIZ (DIMensionada en n)



LISTA (sin longitud determinada)



Listas contra matrices
Las listas ofrecen dos ventajas principales sobre las matrices. En primer lugar, no hace falta reservar memoria para ellas antes de que se la necesite; en segundo lugar, una lista es una estructura "dinámica", lo que equivale a decir que no necesita tener longitud fija y que se puede comprimir o expandir para acomodar datos durante la ejecución del programa. Además, las listas se prestan más fácilmente a los procedimientos recursivos que se utilizan con frecuencia en la programación de inteligencia artificial

Queda detenido

El Z80 ofrece tres modos de interrupción: veamos cómo el sistema operativo del Sinclair Spectrum vectoriza las interrupciones

Las facilidades basadas en interrupciones, propias del sistema operativo del Spectrum, son menos directas que las utilizadas en el BBC Micro. El empleo de las interrupciones en el Spectrum exige sumo cuidado. En caso de inhabilitarlas, o "desactivarlas" inadvertidamente, la máquina puede quedar gravemente afectada: el teclado puede no ser leído, por ejemplo, o puede colgarse el sistema durante la ejecución de programas en BASIC.

La CPU del Z80 es el corazón del Spectrum y responde a dos tipos diferentes de interrupciones: las enmascarables y las no enmascarables (NMI). Su diferencia es clara. Se puede programar la CPU de modo que ignore una señal de interrupción enmascarable, pero el procesador siempre responderá a una NMI (no enmascarable).

Estas interrupciones NMI son poco prácticas en el Spectrum, ya que la rutina ROM que gestiona las NMI muestra bastantes errores. La intención original de los diseñadores del OS era la de permitir al usuario especificar una dirección en las variables de sistema no utilizadas en las posiciones &55cb0 y &5CB1, donde se saltaría cada vez que la CPU recibiera una señal de NMI. Pero la respuesta habitual del Z80 es la ejecución del código máquina que comienza en la dirección &66, lo que se traduce en una inicialización completa del sistema. Por lo tanto, sólo nos ceñiremos al examen de las interrupciones enmascarables.

Modos de operación de interrupción

Varias son las formas en que el Z80 puede responder a una interrupción enmascarable, llamadas precisamente *modos* de operación de interrupción. Trataremos tan sólo aquellos modos importantes para el sistema operativo del Spectrum.

En el momento de la inicialización (que se da al conectar la máquina o al generar la instrucción NEW) la CPU establece el modo de interrupción 1, también conocido por IM1. Éste es el modo habitual en que funciona el Spectrum. Los impulsos de interrupción se llevan a la CPU por medio de la ULA Sinclair (*uncommitted logic array*: tabla lógica sin cometido) a una densidad de 50 interrupciones por segundo. En el modo IM1, la CPU ejecuta una instrucción del RST en &0038 al recibir la señal. Esto provoca un salto a las rutinas que leen el teclado y actualizan el contador FRAMES, situado en la RAM en las posiciones que van de la 23672 a la 23674 (estos tres bytes forman un contador de 24 bits que, por ende, se actualiza cada 20 milisegundos).

Una vez interpretada una línea del BASIC, el intérprete de este lenguaje espera una interrupción antes de pasar a interpretar la siguiente sentencia. Esto significa que si usted neutraliza las interrupciones, la ejecución de un programa en BASIC se parará.

La CPU ignorará una interrupción enmascarable una vez ejecutada la instrucción DI (*Disable Interrupt*: desactivar interrupción), y volverá a tenerlas en cuenta después de la orden EI (*Enable Interrupt*: reactivar interrupción).

Existen diversas rutinas en el OS del Spectrum que exigen la desactivación de interrupciones mientras son ejecutadas. Se trata, por lo general, de rutinas dependientes del tiempo tales como la rutina BEEP del generador de sonido y las rutinas para guardar (o cargar) datos en cinta. Las interrupciones pueden ser desactivadas momentáneamente por medio de la impresora ZX, la Interface 1 o el microdrive. Es claro que el OS vuelve a activarlas una vez acabada la rutina.

Un resultado práctico de las diversas rutinas que desactivan interrupciones es que mientras estas rutinas son ejecutadas el contador FRAMES no se incrementa. Por tanto, se "perderá tiempo" mientras son ejecutadas estas operaciones. Dado que una interrupción puede darse mientras se está ejecutando

Anticiparse a todas las posibilidades

Aunque en muchos casos se puede asumir que el bus de datos del Spectrum contiene, cuando se genera una interrupción, el valor 255, algunos periféricos (como la interface de palanca de mando Kempston) pueden alterar este valor. La forma más fácil de obviar el problema es cerciorarse de que no se está empleando el modo 2 (IM2) con periféricos acoplados. Otro método, abierto a todo valor posible en el bus de datos, consiste en llenar una página de memoria con valores iguales. Se carga después el número de página en el registro I antes de seleccionar el IM2. Cuando se da una interrupción, el Z80 toma la dirección de la rutina de servicio de interrupciones sacándola de alguna posición de la página especificada: la posición exacta que será determinada por el valor que lleve el bus de datos. Primeramente cargamos el registro I con el valor &FC, y llenamos los bytes &FC00 al &FD00 con el valor &FB. Después seleccionamos el IM2. Si el bus de datos contiene &C3 cuando ocurre la interrupción siguiente, el Z80 sacará la dirección de la rutina de servicio de la posición &FCC3: la parte &FC de la dirección ha sido suministrada por el registro I. La dirección tomada será &FBFB, ya que todos los 256 bytes de la página FC (añadiendo el byte cero de la página &FD) han sido llenados con el mismo valor. Hay dos puntos a tener en cuenta. Primero: su rutina de servicio de interrupciones (ISR) tendrá que estar siempre en una dirección con idénticos bytes *hi* y *lo* (p. ej., &C4C4 o bien &FDFD). Segundo: debe recordar que el bus de datos puede que contenga el valor &FF en el momento de tener lugar la interrupción. En este caso el Z80 buscará la rutina ISR en las direcciones nnFF (byte *lo*) y (nn+)00 (byte *hi*), donde nn es el valor del registro I. Por esta razón, debe recordar poner el primer byte de la página siguiente

uno de sus programas, será oportuno que desactive las interrupciones para el fragmento de programa en que la temporización exacta sea importante. Pero, recuerde, es *esencial* que las reactive antes de volver al BASIC.

Interrupciones vectorizadas

Para hacer un uso práctico de las interrupciones del Spectrum, es necesario cambiar el modo y poner aquel que ofrezca mayor versatilidad. El modo de interrupción más útil es el IM2, cuyo funcionamiento es mucho más complicado que el de IM1. Mientras el IM1 siempre consiste en un salto a la dirección &0038 (empleando la instrucción RST &0038), el IM2 puede saltar a cualquier rutina de la memoria. La dirección a la cual salta la CPU se especifica por medio del llamado *vector de interrupción*.

En una interrupción vectorizada, el vector retiene la dirección de la *rutina de servicio de interrupciones* que se ejecutará cuando se dé una interrupción enmascarable. La CPU conoce dónde está situado el vector en la memoria gracias a un registro especial del Z80 llamado *registro I*. La dirección de la rutina del vector de interrupción se halla combinando el contenido del registro I con el del bus de datos en el instante de la interrupción. En algunos sistemas el dispositivo que origina la interrupción colocará un byte en el bus de datos para avisar a la CPU cuál fue la causa de la interrupción.

Pero la ULA del Sinclair no pone ningún valor en el bus de datos cuando envía una señal de interrupción al Z80, aunque la forma en que está configurado el hardware del ordenador permite que el bus de datos contenga el valor 255 cuando no se ha aplicado otra entrada en él. De esta manera, el vector de interrupción siempre estará situado en una frontera de página de memoria, con el byte *lo* de la rutina de servicio contenido en la dirección &nnff y el byte *hi* en la dirección &(nn+1)00, siendo nn el contenido del registro I.

Por ejemplo, si el registro contiene el valor &FB, la dirección del vector estará en &FBFF. El byte *lo* del vector, en la posición &FBFF, contendrá el byte *lo* de la dirección de la mencionada rutina de servicio, y el byte *hi* del vector, en la dirección &FC00 contendrá el byte *hi* de la dirección de la ISR.

Existen algunas reservas sobre el posicionamiento en memoria de la ISR: así, por ejemplo, los primeros 16 Kbytes están asignados a la ROM, y por ello no pueden usarse. Por otro lado, algunos problemas del hardware impiden la operación correcta del OS con un valor del registro I que se comprenda entre el 64 y el 127.

Empleo del modo 2

Examinemos ahora cómo se pone el Z80 en el modo 2, y se establece el vector de interrupción con la dirección de la ISR que ha de emplearse:

```
F3          di
210000      ld    hl,ADDRESS      ;toma dirección ISR en HL
22FFFB      ld    (#FBFF),hl      ;pone dirección en vector
3EFB        ld    a,#FB           ;byte hi del vector...
ED47        ld    i,a             ;...al registro I
ED5E        IM    2               ;establece modo 2
FB          ei                    ;reactiva interrupciones
C9          ret                   ;vuelta al BASIC
```

Naturalmente este listado presupone que hay una rutina en ADDRESS que gestiona las interrupciones; de lo contrario, es posible que todo se venga abajo. Además, toda rutina de servicio de interrupción ejecutará idealmente todas las tareas llevadas a cabo por lo general en el Spectrum en su modo habitual de interrupción. Esto se hace mejor con una llamada a la rutina en la dirección &38. El siguiente programa en lenguaje assembly cambia el modo de interrupción al IM2 y obliga a la CPU a ejecutar la rutina de ADDRESS a cada interrupción (en este caso sólo ejecuta las funciones habituales que realiza el Spectrum en su modo normal).

```
              org   60000           ;especifica dir inicio
      vector: equ   #FEFF           ;FEFF es la dir vector
216FEA change:ld    hl,addres       ;lleva address a HL
22FFFE        ld    (vector), hl    ;establece vector
F3            di                    ;desactiva interrupciones
3EFE          ld    a,#FE           ;establece el...
ED47          ld    i,a             ;...registro I
ED5E          im    2               ;cambia modo interrups.
FB            ei                    ;reactiva interrupciones
C9            ret                   ;vuelta al BASIC
F3      addres: di                  ;desconecta interrupciones
FF            rst   #38             ;procedimiento IM1 normal
FB            ei                    ;reactiva interrupciones
C9            ret
```

Si se llama a la rutina en la dirección CHANGE, se establecerá el nuevo modo de interrupción y el vector. Una vez conseguido esto, la rutina en ADDRESS hará su trabajo cada cincuentavo de segundo. Por el momento esta rutina no hace nada especialmente útil, pero pronto le asignaremos alguna función de interés. Si se necesita cambiar el modo de interrupción al normal en cualquier momento del programa, podemos emplear una rutina como la que sigue para que realice esto:

```
F3          di
3E3F        ld    a,#3f           ;restablece reg I
ED47        ld    i,a             ;...a su valor normal
ED56        im    i               ;modo normal
FB          ei                    ;reactiva interrupciones
C9          ret
```

Para hacer que una rutina de servicio de interrupción ejecute algún código máquina escrito por nosotros, trataremos ese código como una subrutina y lo llamaremos (CALL) de esta manera:

```
F3      addres: di
F5              push af             ;salva registros...
C5              push bc
D5              push de
E5              push hl
FF              rst #38             ;llama ISR normal
F3              di                  ;ISR hizo EI, por tanto DI otra
                                    vez
CD0000          call PROG_ADD       ;llama nuestra rutina
E1              pop hl              ;restaura registros...
D1              pop de
C1              pop bc
F1              pop af
FB              ei
C9              ret
```

Naturalmente, si la rutina de servicio es más bien larga, ralentizará ligeramente al Spectrum, lo que afectará a la ejecución del programa y a la velocidad en que se incrementa FRAMES. Por último, recuerde siempre que debe desactivar las interrupciones durante su rutina para evitar que ocurra una segunda interrupción a mitad del tratamiento de la primera.

Programa "Música bajo interrupciones"

El siguiente programa muestra cómo pueden usarse las interrupciones para obtener "música de fondo" en el Spectrum. El Compilador de Datos de Notas es un programa en BASIC por el que se pueden escribir melodías diversas, mientras que el listado assembly (o el Cargador en BASIC si no se dispone de un ensamblador) le permitirá establecer la rutina musical basada en interrupciones y pasarle los datos compilados por el Compilador de Datos de Notas. Si se ejecuta el Compilador de Datos de Notas se nos presenta un menú con tres opciones. Si optamos por 'C-COMPILE' se convertirá la melodía (contenida en las sentencias data que están entre las líneas 10 y 900) en un fragmento de código máquina utilizable por la rutina de interrupción. La opción 'P' hará sonar la melodía de modo que pueda realizar los cambios que le apetezcan. Mientras 'R' volverá al BASIC para que pueda guardar (SAVE) los datos compilados o cambiar los datos musicales contenidos en las sentencias data. Obsérvese que si se emplea la opción 'C', se obtendrá una interrogación para que especifiquemos dónde se desean almacenar los datos de notas en la memoria. Cerciórese de que los almacena por encima de RAMTOP, cuyo valor habrá alterado previamente para dar espacio a los data. Una vez compiladas las notas y almacenadas, la dirección de base y la longitud del código quedarán visualizados. Para guardar el código, vuelva al BASIC y teclee: SAVE "NOTEDATA" CODE (dirección de base), (longitud en n.º de bytes). Por último, si desea emplear sus propias melodías, elimine las líneas que van de la 10 a la 90 y escriba sus propios datos entre estas líneas. Digite el listado assembly por medio de un ensamblador y guárdelo en cinta. Si no posee un ensamblador emplee el Cargador del BASIC. La rutina tiene una dirección de base en 65021, por lo tanto cerciórese de que antes de cargar el código en la memoria usted ha rebajado convenientemente el RAMTOP (bastará con CLEAR 65000). Una vez ensamblado y cargado el código, será necesario cargar el código compilado por el Compilador de Datos de Notas (ver más abajo). De nuevo no olvide de rebajar el RAMTOP para dar espacio a los datos de las notas. El programa *Música bajo interrupciones* exige el conocimiento exacto de la posición de los datos de notas que ha de hacer sonar, para que funcione. Esta información se puede pasar a la rutina por medio del siguiente programa en BASIC:

```
10 LET L=65152 : LET
   V=(*direccion base de
   los datos de notas*)
20 POKE L+1,INT(V/256):
   POKE L,V-PEEK
   (L+1)*256
```

Hecho esto, RAND USR 65041 establecerá el IM2 y sonará la melodía. RAND USR 65071 seleccionará el IM1 y cesará la música.

Compilador de Datos de Notas

```
  1 >>>MUSICA BAJO INTERRUPCIONES<<<
  2 >>Compilador Datos Notas<<
  3 REM
  6 REM *DATOS DE NOTAS EN FORMA BEEP DEL SPECTRUM
  7 REM
  9 RESTORE
 10 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,1,9,2,10,1,
    10,2,10,1,12,5,14
 20 DATA 1,14,2,7,1,7,1,7,1,6,1,7,2,16,1,14,2,14,1,10,2,9,1,
    9,2,9,1,10,5,12
 30 DATA 1,12,2,9,1,9,1,9,1,8,1,9,2,17,1,12,2,12,1,12,2,11,
    1,19,2,19,1,19,5,19
 40 DATA 1,17,2,16,1,19,1,19,1,18,1,19,2,14,1,19,1,19,1,18,
    1,19,2,12,1,11,2,12,1,11,3,12
 50 DATA 3,10,1,9,1,8,1,9,2,14,1,12,3.2,9,3.2,5,3.2,2,
    2,3.2,7,5,5
 60 DATA 1,5,1,7,1,9,1,10,2,16,1,14,3.2,12,3.2,17,3.2,16,
    3.2,14,5,12
 70 DATA 1,12,2,14,1,14,1,14,1,13,1,14,3,16,3,9
 80 DATA 2,17,1,17,1,19,1,17,1,19,5,21
 90 DATA 1,21,2,19,1,17,2,14,1,10,3.2,9,3.2,5,3.2,7,3.2,4,5,5
997 REM
998 REM NO BORRAR LINEA 999
999 DATA 255,255
1000 REM
2000 REM **** MENU ****
2010 BORDER 1: PAPER 1: INK 6
2020 CLS
2030 PRINT AT 5,4;"COMPILADOR MUSICA";AT 10,5;"C —
     Compile note data";AT 12,5;"P — Play tune in beeps";AT
     14,5;"R — Return to BASIC"
2040 LET a$=INKEY$
2050 IF a$="c" OR a$="C" THEN GO TO 5000
2060 IF a$="p" OR a$="P" THEN GO TO 3000
2070 IF a$="r" OR a$="R" THEN CLS : STOP
2080 GO TO 2040
3000 RESTORE
3010 READ d,f: IF d=255 AND f=255 THEN GO TO 2000
3020 BEEP d/10,f-6: GO TO 3010
```

Listado assembly

```
                org   65021
0000    vect:   defw  0          ;2 bytes para el vector
DDE5    inter:  push  ix
E5              push  hl
C5              push  bc
D5              push  de
F5              push  af
C334FE          jp    start      ;salto a rutina musica
F1      enprg:  pop   af
D1              pop   de
C1              pop   bc
E1              pop   hl
DDE1            pop   ix
C33800          jp    56         ;salto a ISR de ROM
21FFFD  on:     ld    hl,inter   ;establece vector
22FDFD          ld    (vect),hl
F3              di
3EFD            ld    a,253
ED47            ld    i,a        ;toma byte hi de vect en I
ED5E            im    2          ;pone modo 2 interr
2A80FE  initi:  ld    hl,(datad) ;apunta a data
227EFE          ld    (locat),hl
3E01            ld    a,1        ;switch=1=suena nota
3282FE          ld    (swtch),a
3D              dec   a          ;demora=0
3283FE          ld    (delay),a
FB              ei
C9              ret              ;vuelta al BASIC
F3      off:    di
ED56            im    1          ;pone modo 1 interr
FB              ei
                ret
C9      ;
3A82FE  start:  ld    a,(swtch)  ;toma switch en a
```

```
5000 RESTORE : CLS : INPUT "DIRECCIONE BASE DE DATA
     NOTAS? ";dd1: POKE 23301, INT (dd1/256): POKE
     23300,dd1-(256*PEEK 23301): CLEAR dd1-1: LET
     dd1=PEEK 23300+256*PEEK 23301
5005 LET f=0: LET d=dd1
5010 READ delay,pitch
5015 LET delay=INT (delay*6)
5020 LET freq=(1.0594631^pitch)*256
5030 LET bip=INT ((437500/freq)-30.125)
5035 IF delay=255*6 THEN POKE d,255: GO TO 6000
5037 LET f=f+1: PRINT AT 10,10;"NOTA)";f
5040 POKE d,delay
5060 POKE d+1,bip-(INT (bip/256)*256)
5070 POKE d+2,INT (bip/256)
5080 LET d=d+3
5090 GO TO 5010
6000 PAUSE 50: CLS : LET len=(d+2)-dd1
6010 PRINT '""NUMERO DE BYTES DE DATA DE NOTAS...";len
6020 INPUT "PULSE 'ENTER' PARA VOLVER A MENU "; LINE a$:
     GO TO 2000
```

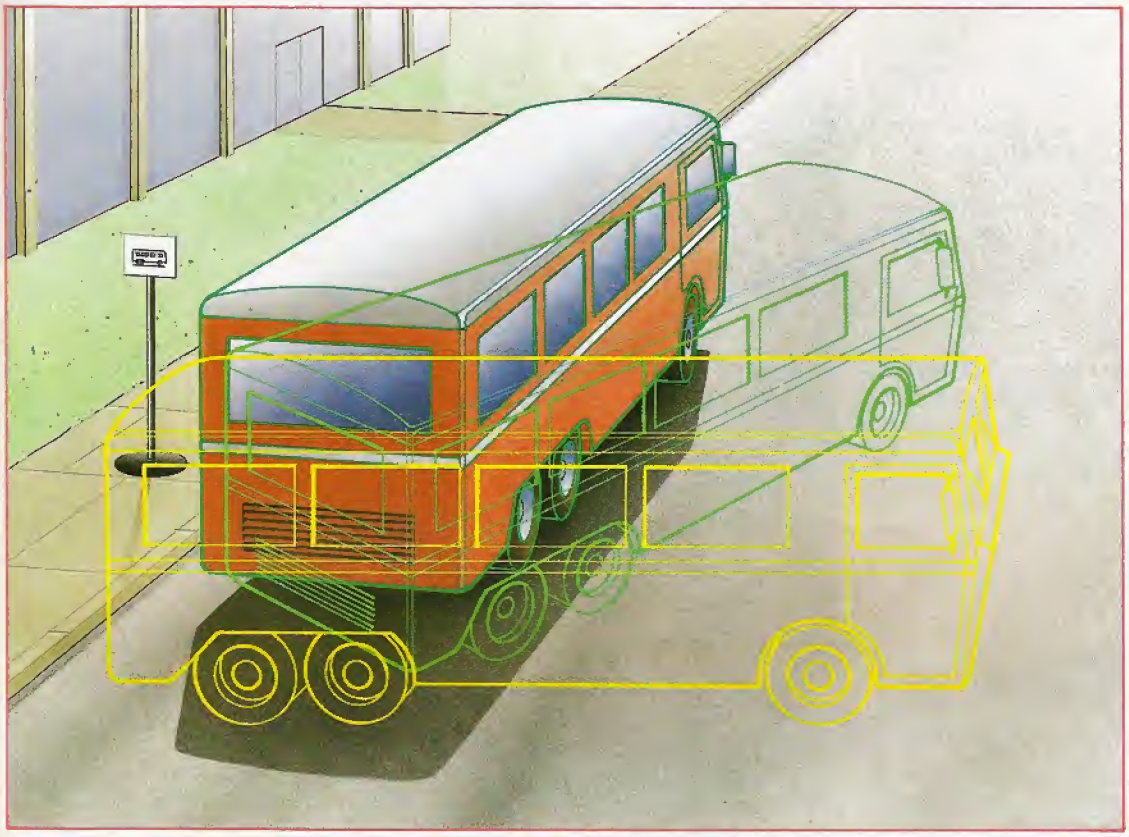
Cargador del BASIC

```
  10 RESTORE 9000
  20 CLEAR 64000
  30 LET cqs=0: FOR f=65021 TO 65156
  40 READ dat: POKE f,dat
  50 LET cqs=cqs+dat
  60 NEXT f
  70 IF cqs<>18850 THEN PRINT "ERROR EN DATA!!!" : STOP
  80 PRINT "OK CORRECTOS LOS DATA"
  90 STOP
9000 DATA 0,0,221,229,229,197,213,245,195,52,254,241,209,
     193,225,221,225,195,56,0,33,255,253,34,253,253,243,
     62,253,237,71,237,94,42,128,254,34,126,254,62,1,
     50,130,254,61,50,131,254,251,201,243,237,86,251,201,
     58,130,254,254,0,202,8,254,58,131,254,254,0,
     202,75,254,61,50,131,254,195,8,254,42,126,254,126,
     254,255,202,108,254,50,131,254,35,126,35,95,126,35,
     34,126,254,103,107,17,4,0,205,181,3,243,195,8,
     254,42,128,254,34,126,254,62,1,50,130,254,61,
     50,131,254,195,8,254,0,0,0,0,0,0,0,
```

```
FE00            cp    0
CA08FE          jp    z,enprg    ;si switch=0,fin
3A83FE          ld    a,(delay)
FE00            cp    0
CA4BFE          jp    z,nwnot    ;si demora=0,sonar
3D              dec   a          ;demora=demora-1
3283FE          ld    (delay),a  ;almac. nuevo valor dem.
C308FE          jp    enprg      ;ir a rutina fin
2A7EFE  nwnot:  ld    hl,(locat) ;toma posicion datos
7E              ld    a,(hl)     ;toma demora sig. en a
FEFF            cp    255        ;255=datos finalizados
CA6CFE          jp    z,reset    ;datos finalizados, restaurar
3283FE          ld    (delay),a  ;almacena nueva demora
23              inc   hl         ;apunta a sig. byte datos
7E              ld    a,(hl)     ;lo toma en a
23              inc   hl         ;apunta a sig. byte datos
5F              ld    e,a        ;almacena byte 1 en e
7E              ld    a,(hl)     ;almacena byte2 en a
23              inc   hi         ;apunta a byte sig.
227EFE          ld    (locat),hl ;almacena dirección byte sig.
67              ld    h,a        ;toma datos frecuencia...
68              ld    l,e        ;...en HL
110400          ld    de,4       ;duración nota de 4
CDB503          call  949        ;llama rut. BEEP en ROM
F3              di               ;esta rutina hizo EI, por tanto DI
C308FE          jp    enprg      ;ir a rutina fin
2A80FE  reset:  ld    hl,(datad) ;restaura varios...
227EFE          ld    (locat),hl ;...punteros
3E01            ld    a,1
3282FE          ld    (swtch),a
3D              dec   a
3283FE          ld    (delay),a
C308FE          jp    enprg      ;ir a rutina fin
0000    locat:  defw  0          ;reserva memoria...
0000    datad:  defw  0          ;...para varios punteros
00      swtch:  defb  0
00      delay:  defb  0
```

Conocimiento visual

Al programar un ordenador para que «comprenda» lo que ve, es necesario diferenciar lo que se ve de lo que se conoce



Kevin Jones

Vista de lince
Algunos sistemas de reconocimiento de patrones adoptan un enfoque *de arriba-abajo*, en el que se explora un patrón o escena en busca de la presencia de un objeto determinado. Sobre la pantalla se proyecta una representación esquemática del objeto en diversas orientaciones hasta que, si el objeto está presente, se encuentra una proyección que encaja en la silueta de la figura real. En una escena tridimensional, como la que vemos aquí, es muy grande la cantidad de posibles proyecciones que se podrían hacer de un modelo esquemático del autobús antes de hallar la correcta

Dicen que «la belleza está en los ojos del que mira»; pero sería mejor decir que la belleza está en el cerebro del perceptor o, con mayor precisión, en una compleja cadena de procesos nerviosos que comienza en el fondo de la retina y termina en algún lugar de la corteza occipital del cerebro. Es esta cadena de eventos lo que los fisiólogos intentan comprender y lo que los expertos en robótica intentan, en cierto sentido, imitar.

La percepción visual es un componente tan integral de la forma en que comprendemos nuestro entorno, que con frecuencia decimos «Ya veo» cuando en realidad queremos significar «Ya comprendo». La comprensión es realmente la clave del desafío que representa la visión por ordenador. De alguna forma, el ordenador debe tomar la información que le proporciona una cámara, u otro dispositivo sensible a la luz, y entender lo que esa información le dice sobre el estado de su entorno. Adquirir la información es fácil: el problema es cómo interpretarla.

El proceso de transformar imágenes en significados consta de tres etapas principales:

1. *Proceso de la imagen:* Convertir una imagen difusa o distorsionada en una más nítida. (La solución de esta tarea es relativamente simple.)
2. *Reconocimiento de patrones:* Detectar la presencia o la ausencia de rasgos u objetos significativos. (Esto es más complicado.)
3. *Comprensión de la imagen:* Determinar lo que está sucediendo en el mundo real. (Un problema muy complejo.)

Aunque la auténtica visión por ordenador (etapa 3) todavía no se ha alcanzado, de las dos etapas iniciales sí se han obtenido resultados útiles.

Reconocimiento de patrones

El reconocimiento de patrones es una tarea fascinante, si bien nos resulta difícil «ver» las dificultades que entraña. Los reconocedores de patrones clasifican las imágenes emparejándolas con un conjunto limitado de alternativas, tales como las letras del alfabeto en el caso de los sistemas de reconocimiento de caracteres ópticos. Típicamente, extraen características mediante el examen de partes de la imagen digitalizada (pequeños grupos de pixels). El sistema puede entonces reconocer y clasificar patrones basados en la presencia o la ausencia de ciertas características distintivas.

El reconocimiento de algunas características, como pueden ser los trazos en diagonal, para nosotros es claro, pero con frecuencia la identificación de tales sistemas puede resultar puramente de

cálculos arbitrarios efectuados sobre los datos de la imagen. Algunos sistemas poseen importantes usos prácticos, pero apenas si tocan la superficie de lo que entendemos por visión.

Históricamente, la visión por ordenador como comprensión de la imagen ha tenido dos enfoques opuestos: el enfoque *de arriba-abajo* o activado por modelos, y el enfoque *de abajo-arriba* o activado por datos.

El análisis de escenas activado por modelos con frecuencia se denomina *alucinación controlada*. El sistema posee un modelo interno de lo que está buscando (un autobús de dos pisos, p. ej.) y proyecta ese modelo sobre el plano de imagen en diversas orientaciones. Luego comprueba la calidad de encaje entre lo que ha «imaginado» y lo que hay verdaderamente allí.

Los sistemas de abajo-arriba exploran los datos de la imagen en busca de líneas, bordes y otros signos reveladores. Mediante esa exploración construyen una descripción simplificada de la imagen, que se denomina *boceto primitivo* y es una representación muy abstracta de los datos. La idea es que al prescindir de gran parte de los detalles se hace más fácil comparar el boceto con ejemplos tomados de una categoría dada de objetos almacenados como plantillas de referencia.

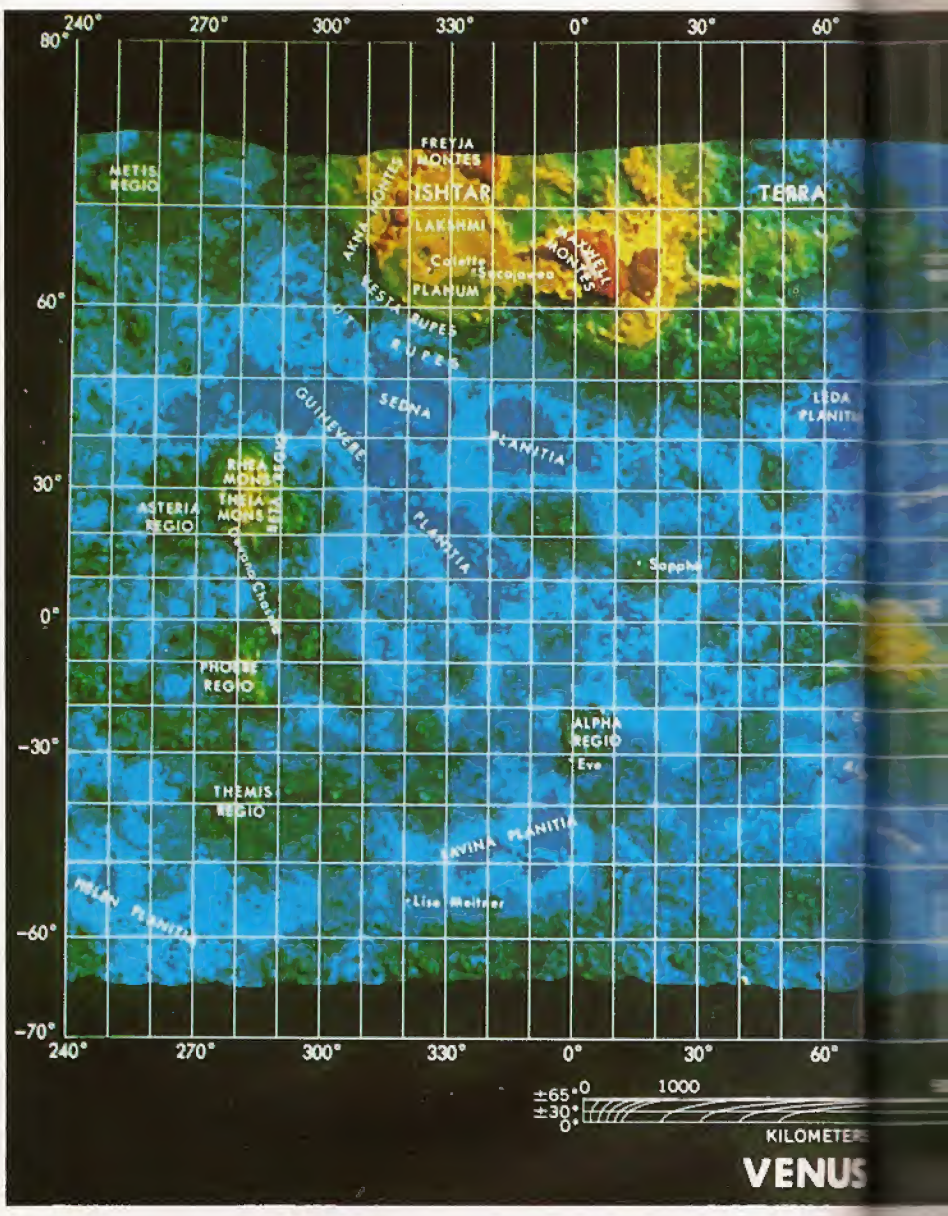
Los sistemas prácticos tienden a emplear tanto el método de arriba-abajo como el de abajo-arriba, pero hasta muy recientemente había muy pocos sistemas de aprendizaje en el campo de la visión por ordenador. Todos ellos dependían de inteligencia preprogramada.

Una excepción notable es el WISARD (Wilkie, Aleksander & Stonham's Recognition Device) de Igor Aleksander. Se le puede enseñar a hacer la distinción, comparativamente sutil, entre un rostro humano sonriente y uno con el ceño fruncido. Puede incluso generalizar este conocimiento a rostros que no haya visto nunca antes. El WISARD opera en tiempo real (25 fotogramas por segundo) y su éxito comercial para clasificar chocolates sobre una cinta transportadora móvil podría allanar el camino para una nueva generación de «máquinas que ven» que adapten y mejoren su rendimiento.

De forma muy resumida, el WISARD trabaja asignando grupos de ocho pixels por vez (óctuplos) a bancos seleccionados de RAM. Un óctuplo es un detector de rasgos y puede estar en uno de 256 estados (de 0 a 255), según el estado de cada uno de los pixels que controla. Durante la fase de entrenamiento, se almacena un 1 en la posición del banco de RAM especificada por el estado del óctuplo cuando hay presente una imagen determinada. Luego, en la fase de reconocimiento, la presencia de un 1 en la posición direccionada del banco de RAM de ese óctuplo constituye una evidencia de que la imagen «enseñada» está otra vez presente.

Mediante el empleo de una gran cantidad de óctuplos, o discriminadores, el sistema se vuelve relativamente impermeable a datos espurios (imágenes «ruidosas»).

El sistema de Aleksander posee una cuadrícula de 512 por 512 para la imagen y más de 32 000 óctuplos. Ello requiere alrededor de ocho millones de bits (un megabyte) de RAM. Tales memorias se han vuelto económicamente viables desde hace muy poco tiempo.



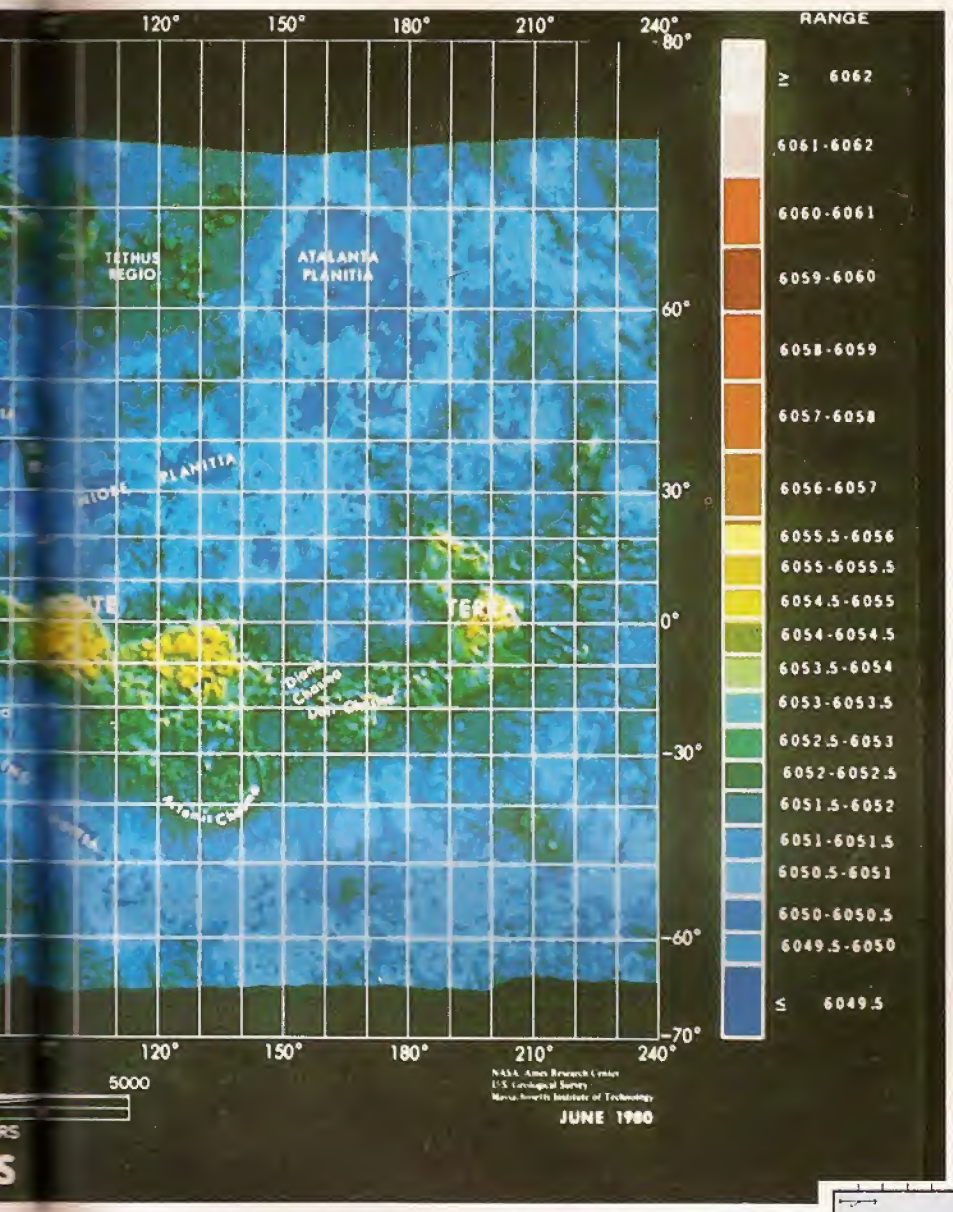
0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	1	1	0	0
0	1	1	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	0	0	0	1	0

0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0
0	0	1	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	1	1	1	0	0	0
0	1	1	0	0	1	0	0
0	1	0	0	0	1	0	0
0	1	0	0	0	0	1	0

Caroline Clayton

La diferencia entre R y A

En el mundo real, los datos bien definidos son muy poco frecuentes. En el ejemplo de arriba vemos formas digitalizadas de las letras A y R, pero ningún patrón está bien definido, al existir manchas y distorsiones. Los datos «ruidosos» como éstos son un problema constante para todos los tipos de sistemas de inteligencia artificial. ¿Podría usted determinar cuál es cuál?

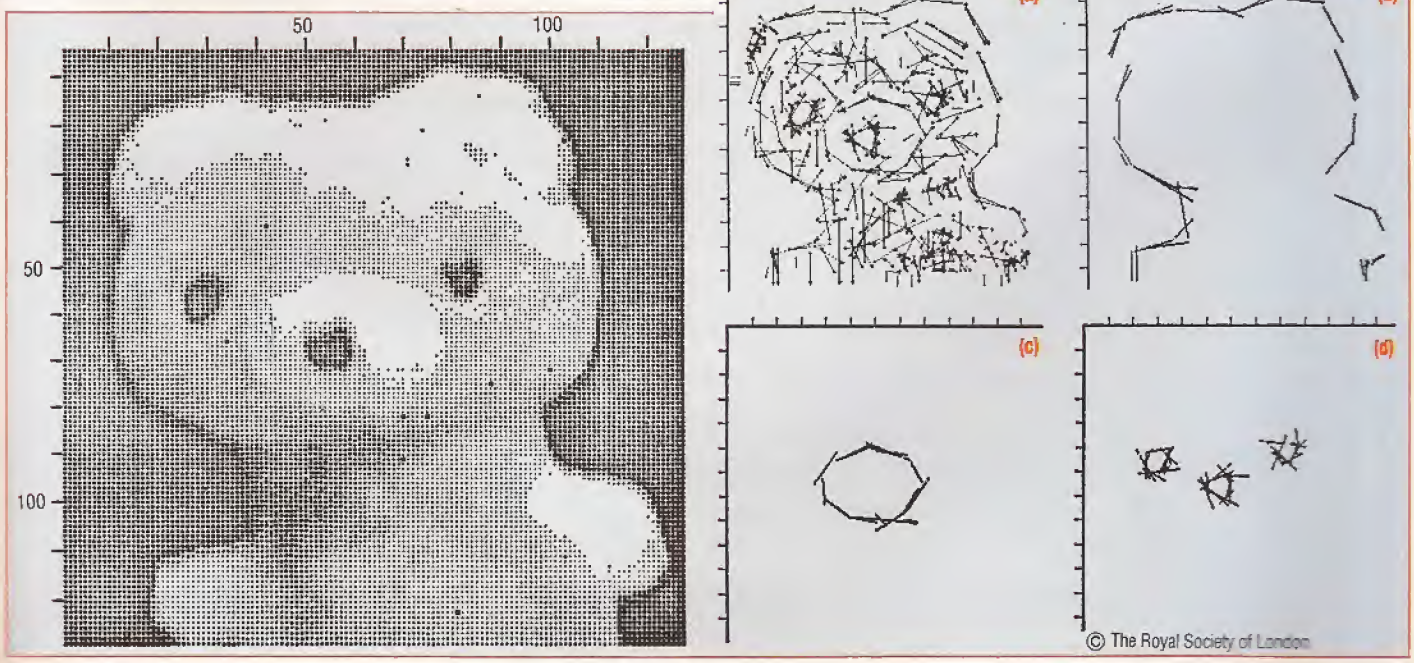


Proceso de imágenes

Ilustramos un ejemplo de proceso de imágenes. Aquí se ha procesado por ordenador una fotografía tomada desde una sonda espacial, para acentuar las fronteras entre regiones de distinto relieve. Ello implica el uso de *colores falsos*. Lo interesante de este grabado, que en realidad es una imagen por radar del planeta Venus, es que todos los colores son «falsos». Sin la ayuda del ordenador jamás podríamos ver esta imagen, porque Venus está envuelto en densas nubes y nuestros ojos no son sensibles a las longitudes de onda que se requieren para penetrarlas. En astronomía, tales imágenes han resultado muy útiles (y algunas veces muy bellas) en los últimos años, aprovechando, además de la luz visible, las longitudes de onda de radio, infrarrojas y ultravioletas. En nuestro planeta se está investigando el lecho oceánico por sonar para construir «imágenes acústicas» que emplean el principio del sondeo acústico y no dependen en absoluto de la radiación electromagnética (luz visible, ultravioleta, infrarroja, rayos X, etc.). Otras clases de proceso de imágenes se utilizan para «limpiar» imágenes borrosas o distorsionadas. Tales algoritmos se emplean ahora de forma tan rutinaria que ya no se consideran componentes de AI

Bocetos primitivos

Se ha desarrollado un método de proceso visual que no depende del conocimiento previo de lo que se supone que representa el patrón. El boceto primario (a) extrae de la imagen del osito de peluche fronteras y formas primitivas, mediante la comparación de regiones adyacentes. Sucesivas extracciones del boceto primitivo (de [b] a [d]) revelan grupos importantes que contribuyen a reconocer la imagen original



Diseño funcional

Continuando con nuestra serie dedicada al LISP, examinamos sus dos conceptos fundamentales: listas y funciones

En el primer capítulo de esta serie vimos cómo informar al LISP de que queremos que una lista determinada se evalúe como una lista de elementos de datos y no como una función. De modo que:

```
(SETQ X '(A B C D E F))
```

asignaría la lista (A B C D E F) a la variable X, donde cada elemento de la lista es una serie de un único carácter. Veamos lo que sucedería si D, E y F fueran series de un único carácter y A, B y C fueran variables con los valores 2, 4 y 8, respectivamente. En este caso, en realidad desearíamos asignar a X la lista (2 4 8 D E F). Podemos hacerlo introduciendo una nueva función: LIST. Ésta crea una lista de sus argumentos. Por lo tanto:

```
(SETQ X (LIST 'A 'B 'C 'D 'E 'F))
```

sería exactamente lo mismo que la expresión previa, pero:

```
(SETQ X (LIST A B C 'D 'E 'F))
```

asignaría correctamente la lista (2 4 8 'D 'E 'F). Aquí, al omitir los apóstrofos precedentes, se evaluarían A, B y C. De forma muy similar:

```
(SETQ X (LIST 1 2 4 '(PLUS 4 4))
```

asignaría a X la lista (1 2 4 (PLUS 4 4)), donde el cuarto elemento de la lista es en sí mismo una lista compuesta de los tres elementos PLUS, 4 y 4. En otras palabras, PLUS no se ha evaluado porque hemos precedido la expresión con un apóstrofo. Sin embargo:

```
(SETQ X (LIST 1 2 4 (PLUS 4 4))
```

asignaría a X la lista (1 2 4 8), donde PLUS se ha evaluado (lo que demuestra la diferencia que puede suponer un simple apóstrofo). Podemos ampliar aún más este concepto para crear una lista de listas.

Por ejemplo:

```
(SETQ PERSONA' ((JUAN PI) (17 1 1960)))
```

asignaría a la variable PERSONA una lista de datos.

La lista contiene dos elementos, siendo cada uno de ellos una lista. La primera contiene dos elementos de caracteres y la segunda, tres elementos numéricos. Usted puede seguir ampliándola para obtener todavía más listas de listas.

Hasta ahora hemos visto las funciones SETQ, LIST, PLUS y TIMES. El LISP tiene muchas más funciones incorporadas, y su utilización dependerá de cada implementación en particular. Además, usted puede definir sus propias funciones; de hecho ésta es la forma en que se construyen los programas en LISP.

Antes de ver cómo se hace esto, examinemos las tres funciones básicas que existen en todas las implementaciones de LISP. Éstas son CAR, CDR y CONS. El nombre CONS alude simplemente a CONStrucción. Los nombres CAR y CDR datan de una de las implementaciones originales del lenguaje, y corresponden a las siglas de *Contents of Address Register* (contenido del registro de direcciones) y *Contents of Decrement Register* (contenido del registro de decremento), respectivamente. Tanto CAR como CDR toman un único argumento, que ha de ser una lista con al menos un elemento. Esto excluye la estructura de lista especial conocida como *lista vacía,* o NIL, que se escribe:

```
( )
```

La función CAR devuelve como resultado el primer elemento de su argumento. Así:

```
(CAR '(1 2 3 4 5))
```

devolvería el valor entero 1. No hay ningún motivo por el cual el resultado no pueda ser en sí mismo una lista.

Por ejemplo, en la expresión:

```
(CAR '((1 2) (3 4) 5))
```

el primer elemento es la lista (1 2).

La función CDR efectivamente devuelve todo menos el primer elemento de una lista. Dicho en otras palabras, devuelve lo que queda tras una operación CAR. De modo que:

```
(CDR '(1 2 3 4 5))
```

devolvería la lista (2 3 4 5), y:

```
(CDR '((1 2) (3 4) 5))
```

devolvería la lista ((3 4) 5).

CONS espera dos argumentos y los concatena entre sí, añadiendo el primer argumento a la lista del segundo argumento.

Por lo tanto:

```
(CONS 1' (2 3 4 5))
```

construiría la lista (1 2 3 4 5) y:

```
(CONS '1 2) '((3 4) 5))
```

construiría la lista ((1 2) (3 4) 5).

A menudo las funciones CAR y CDR se encontrarán anidadas, de modo que:

```
(CDR '((1 2) (3 4) 5))
```

devolverá ((3 4) 5), y:

```
(CAR (CDR '((1 2) (3 4) 5)))
```

devolverá (3 4), que es (CAR '((3 4) 5)), y:

```
(CAR (CAR (CDR '((1 2) (3 4) 5))))
```

devolverá el valor 3, que es (CAR '(3 4)).

Esto enseguida resulta tedioso, de modo que el LISP suele admitir abreviaciones. Los nombres de las funciones comienzan igualmente con C y terminan con R, pero pueden tener entremedio cualquier combinación de Aes (para CAR) y Des (para CDR). Utilizando el LISP Acornsoft, esta

combinación puede constar de hasta tres letras empotradas. Por tanto, podríamos escribir la última expresión así:

```
(CAAR (CDR '((1 2) (3 4) 5)))
```

o:

```
(CAR (CADR '((1 2) (3 4) 5)))
```

o incluso:

```
(CAADR '((1 2) (3 4) 5))
```

todas las cuales darían como respuesta 3.

Definición de funciones

Como ya hemos observado, los programas en LISP se construyen como una serie de funciones definidas por el usuario. Hemos visto, además, que todas las expresiones son listas de funciones, de modo que es bastante natural que utilicemos funciones para definir funciones. En este caso, empleamos la función DEFUN, que espera tres argumentos de la forma:

```
(DEFUN a (b)(c))
```

donde a es el nombre de la función, b es la lista de parámetros (como las del PASCAL, el BASIC BBC, etc.) y c es una estructura de lista que contiene el cuerpo principal de la función. Estamos ahora en condiciones de definir nuestra primera función. Supongamos que frecuentemente deseáramos multiplicar números por el valor 8. Podríamos simplemente escribir:

```
(TIMES 8 N)
```

cada vez, donde N es el número a multiplicar. En cambio, vamos a definir una función para hacer lo mismo:

```
(DEFUN TIME8 (N) (TIMES 8 N))
```

Aquí la función TIME8 toma su argumento N y utiliza la función estándar TIMES para multiplicar este número por 8 y devolver un valor.

De modo que la expresión:

```
(TIME8 11)
```

ahora daría como resultado 88.

Antes de que prosigamos, hemos de examinar un importante concepto de programación: la condición. ¡Su importancia puede quedar clara si pensamos en lo que sería el BASIC sin sentencias IF...THEN!

La sentencia condicional del LISP asume la forma de una función:

```
(COND  (Condición1 Expresión1)
       (Condición2 Expresión2)
              :          :
       (CondiciónX ExpresiónX))
```

En primer lugar, observe que esta función se ha extendido a varias líneas para darles cabida a todas.

Esto es bastante normal en LISP, que sabe cuándo usted ha terminado una expresión porque habrá cerrado el paréntesis final.

La función COND es muy parecida a la sentencia CASE del PASCAL. Se evalúa la Condición1 y, de ser verdadera, entonces se utilizará la Expresión1 para devolver un resultado. De lo contrario, se comprobará la Condición2, y así sucesivamente. Algunas implementaciones de LISP (incluyendo el LISP Acornsoft) exigen que la evaluación de al menos una de las condiciones (denominadas *predicados*) resulte verdadera. Esto se maneja fácilmente mediante la adición de una condición final de la forma:

```
(COND  (Condición1 Expresión1)
       (Condición2 Expresión2)
              :          :
       (T              Expresión final))
```

Aquí el carácter T se utiliza para representar *True*; de modo que la expresión final (en caso de que se llegue a ella) siempre se evaluará. En LISP:

```
T = True = No cero
```

y:

```
F = False = Cero = (  )
```

Lo último es la lista vacía. Ahora podemos definir una función algo más compleja. Una facilidad muy útil de la que disponen la mayor parte de las versiones de BASIC es ABS, que devuelve el valor absoluto de su argumento. En otras palabras, si su argumento es positivo, quedará sin modificar; de lo contrario, será negado.

Nuestra función de LISP habrá de ser de la forma:

```
(DEFUN ABS (X)
   (cuerpo de la función))
```

donde X es el argumento entero. Empleando nuestra nueva función condicional, podemos escribir toda la función ABS como:

```
(DEFUN ABS (X)
   (COND ((MINUSP X) (MINUS X))
         (T          X          )))
```

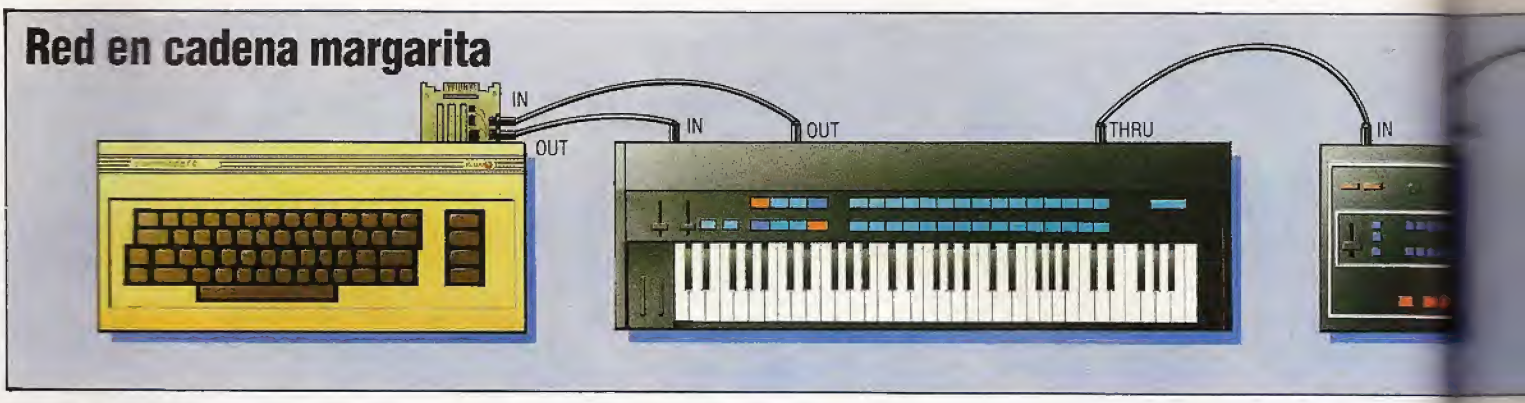
Aquí hemos utilizado dos nuevas funciones. MINUSP es una función de comprobación que devuelve el valor si su argumento es un número negativo, y falso de lo contrario. Si la condición evalúa a T, la función MINUS cambia su argumento a negativo. Si no es éste el caso, la condición T (que siempre es verdadera) devolverá el valor X original.

```
(DEFUN EQUAL (A B) (COND
  ((EQ A B) T)
  ((OR (ATOM A) (ATOM B)) NIL)
  ((EQUAL (COCHE A) (COCHE B)) (EQUAL (CDR A) (CDR B)))
  (T NIL)))
(DEFUN HALLAR_COCHE (X L) (COND
  ((NULL L) '(NINGUN NUMERO))
  ((EQUAL X (CDAR L)) (COCHE L))
  (T (HALLAR_COCHE X (CDR L)))))
(SETQ COCHES '((PEREZ ABC123X) (TORRES XYZ789Y)
          (MARTIN ACE99L) (LUCIA JBB1))
(PRINTC (HALLAR_COCHE '(ACE99L) COCHES))
```

Procedimiento de búsqueda
Este sencillo programa de recuperación de datos demuestra el uso de las funciones CDR y EQ. El número de la matrícula de un automóvil, entrado por el usuario, devolverá, de estar incluido en la base de datos del programa, una lista de dos elementos: el número de matrícula y el apellido del propietario del coche

Funciones de comprobación
He aquí una lista de otras funciones de comprobación que se suelen utilizar en LISP:

(OR arg 1 arg 2...)	Devuelve True si al menos uno de sus args. es verd.
(ONEP arg)	Devuelve True si arg=1
(ZEROP arg)	Devuelve True si arg=0
(LISTP arg)	Devuelve True si arg es una lista y False si es un átomo
(ATOM arg)	Devuelve True si arg es un átomo y False si es una lista
(LESSP arg1 arg2)	Devuelve True si arg1 < arg 2
(GREATERP arg1 arg2)	Devuelve True si arg1 > arg 2
(EQ arg1 arg2)	Devuelve True si arg1 = arg2



Margarita, Margarita...
El método más usual para conectar instrumentos compatibles con la MIDI es mediante una cadena margarita. La mayoría de los instrumentos poseen conectores MIDI THRU (además de MIDI IN y MIDI OUT). Éstos, en efecto, permiten la formación de un sistema tipo bus, en el cual todos los instrumentos están conectados al bus pero sólo responden a mensajes por canales determinados

Reglas de composición

Construida ya la placa de la interface, concentraremos nuestra atención en el software

Ya hemos visto cómo se transmite un byte de datos a través de la MIDI hasta un instrumento receptor. Ahora vamos a determinar el formato del (los) byte(s) necesarios para comunicar la información requerida. Un único «evento» musical se transmite como un grupo de bytes denominado *mensaje*. La longitud de la mayoría de los mensajes está entre uno y tres bytes, con la excepción de los mensajes «exclusivos para el sistema» (que analizaremos más adelante), cuya longitud puede ser de cualquier número de bytes. Cada mensaje comienza con un byte cuyo bit más significativo (MSB) es igual a uno, seguido por el resto de los bytes del mensaje, todos los cuales tienen sus MSB establecidos en cero. Se dice que un byte con su MSB establecido en uno es un byte de *estado*; los otros son bytes de *datos*. Los mensajes MIDI se dividen en dos tipos básicos: *canales* y *sistemas*.

La necesidad de los *mensajes de canal* surge del «encadenamiento en margarita», el método de interconexión más usual utilizado en sistemas pequeños. En estos sistemas, cada unidad recibe todos los datos enviados por la unidad transmisora maestra. Los mensajes de canal se utilizan para transmitir información que no esté dirigida necesariamente a todas las unidades de un sistema. Por consiguiente, a la mayoría de los receptores MIDI se les puede asignar un *número de canal* entre 1 y 16.

Los mensajes de canal poseen bytes de estado con valores comprendidos en la escala entre $80 y $EF inclusive, y tienen uno o dos bytes de datos. El número de canal está codificado en los cuatro bits menos significativos (el dígito hexa menos significativo) del byte de estado, representando $0 el canal 1 y $F el canal 16. Los tres bits restantes determinan el tipo de mensaje que viene a continuación.

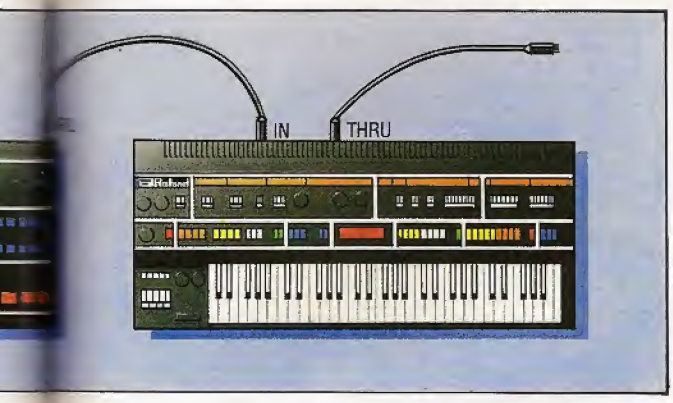
Una característica especial de los mensajes de canal es que no es necesario enviar el byte de estado si se trata del mismo que el byte de estado anterior. En efecto, el estado actual o *corriente* permanece vigente hasta que se recibe otro byte de estado. La excepción es cuando un mensaje de sistema en tiempo real interrumpe temporalmente el estado actual.

Esto es particularmente útil para comunicar muchos mensajes consecutivos de nota *on/off*, dado que un mensaje *nota on* con una velocidad (con cuánta fuerza se pulsa una tecla) de cero equivale a un mensaje *nota off*. La utilización del mismo estado (*nota on*) para una serie de mensajes de nota *on/off* resulta en un ahorro de espacio de memoria y de tiempo de transmisión.

En los sintetizadores, *asignación de voz* es el proceso de dirigir un mensaje de nota (ya sea desde la MIDI o desde el teclado del instrumento) a una de las voces del sintetizador disponibles en ese momento para producir realmente la nota. Por ejemplo, se dice que un sintetizador polifónico de seis notas posee seis voces. Para controlar la respuesta del instrumento a los mensajes de canal, los *mensajes de modalidad* MIDI pueden seleccionar cierto número de modalidades. El receptor que no sea capaz de operar en la modalidad requerida ignorará el mensaje de modalidad.

La *modalidad omni* es la más simple; en ella el receptor responde a todos los mensajes de canal con independencia del número de canal MIDI codificado en los mensajes. Cuando se apaga la modalidad omni, la unidad responde sólo a los mensajes enviados por canales MIDI específicos.

La mayoría de los sintetizadores polifónicos poseen un número de voces limitado (por lo general, seis u ocho), lo que significa que debe emprenderse alguna acción si se requiere una nota cuando ya han sido asignadas todas las voces por mensajes de nota anteriores. Por lo general las voces se asignan por estricto orden de rotación, de modo que la nota menos reciente se apaga para que la voz quede disponible para la nueva nota. Esta modalidad de operación se selecciona mediante el mensaje *poly*



mode on. Un mensaje *mono mode on* indica al receptor que asigne cada una de sus voces monofónicamente a uno de un grupo de canales MIDI consecutivos comenzando desde el canal original (básico) del sintetizador. El mensaje de modalidad mono se envía por el canal básico y su segundo byte de datos especifica el número total de canales requeridos.

Si está activada la modalidad omni, un mensaje mono simplemente indica al receptor que asigne monofónicamente una voz a los mensajes de canal MIDI en todos los canales.

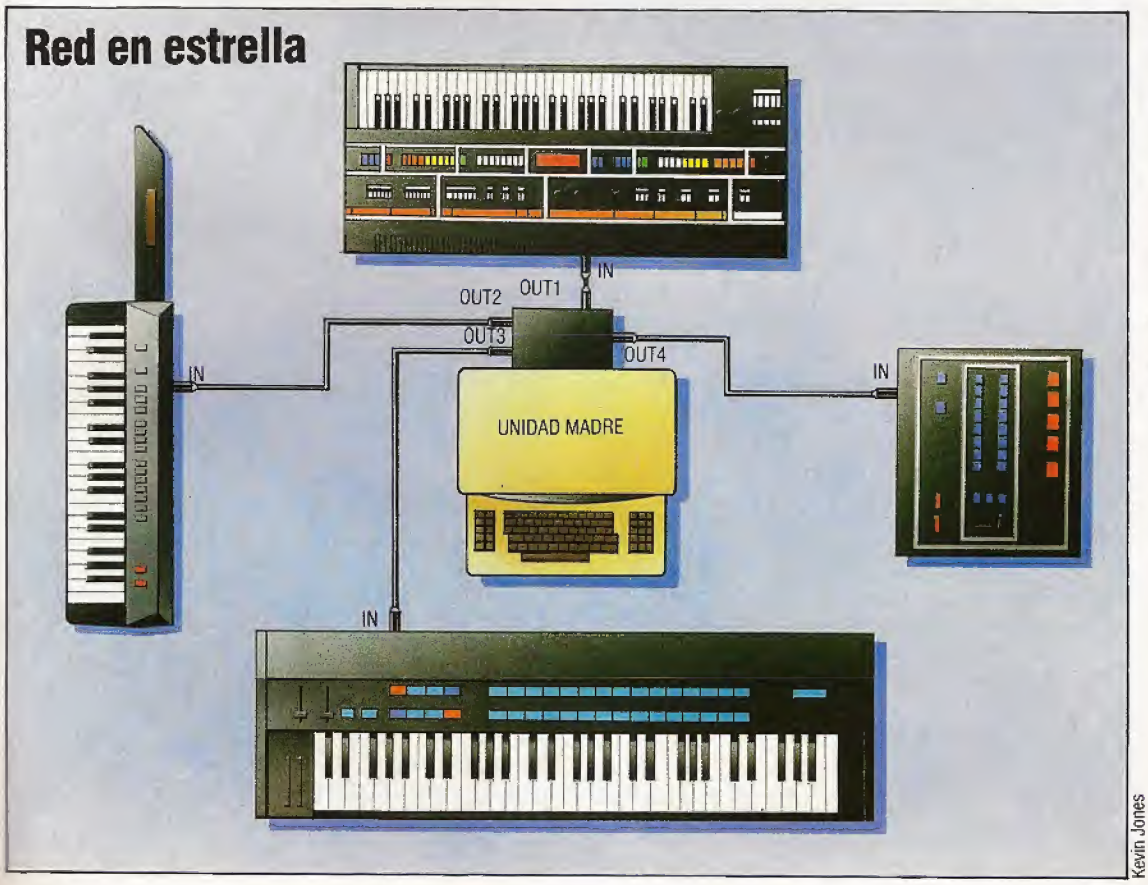
Los *mensajes del sistema* no se codifican con números de canal en sus bytes de estado. Por consiguiente, todas las unidades del sistema reciben el byte de estado. Los mensajes del sistema caen en tres categorías: comunes, en tiempo real y exclusivos.

Los mensajes de *sistema comunes* poseen bytes de estado desde $F1 hasta $F7. Se componen por el byte de estado seguido de 0, 1 o 2 bytes de datos.

Los mensajes del *sistema en tiempo real* están dirigidos a todas las unidades del sistema. Tienen bytes de estado desde $F8 hasta $FF y los utilizan principalmente las máquinas de ritmos y los secuenciadores para sincronizar sus propias secuencias internas con el reloj del transmisor maestro. La mayoría de los sintetizadores ignorarán estos mensajes a menos que posean alguna forma de secuenciador interno con capacidad para sincronización MIDI.

Los mensajes del sistema en tiempo real difieren de los otros mensajes en que se componen de un byte de estado solamente y ningún byte de datos. En consecuencia, se permite enviarlos en cualquier momento, aun cuando interrumpan mensajes de otros tipos.

Los mensajes *exclusivos del sistema* comienzan con el byte de estado $F0, seguido por cualquier número de bytes de datos, y acaban ya sea con el byte de estado de final de exclusivo ($F7) o bien con cualquier otro byte de estado. El primer byte de datos es el código de identificación (ID) del fabricante. Si éste no coincide con el ID de la unidad receptora, se ignorará el resto del mensaje. Los mensajes están destinados a la transferencia de datos entre instrumentos de tipo similar, que carecerían de significado para otras unidades. El principal tipo de datos transferido son los datos de *programa parche* para sintetizadores. (No debe confundirse la transmisión de un programa parche de esta forma con el mensaje de canal $Cx, que selecciona entre programas parche ya almacenados en la memoria del receptor.) Sin embargo, el tipo de datos transmitido por los mensajes exclusivos del sistema queda en manos del fabricante, siempre y cuando se le haya asignado un código ID válido.



Kevin Jones

Una actuación estelar
Las primeras implementaciones de la MIDI no permitían la selección de canales, de modo que todos los sintetizadores intentaban responder a todos los canales, con lo cual el sistema resultaba casi inútil cuando tales dispositivos estaban encadenados en margarita. Estas primeras unidades MIDI habían de estar conectadas en una formación de estrella alrededor de una unidad madre, que incorporaba un sistema de circuitos de modo tal que pudiera obtenerse cada canal desde un conector OUT separado

Mensajes de canal de voz

Estado	Data1	Data2	Descripción	Notas
$8x	p	v	Nota apagada p=altura v=velocidad fuera	1,2
$9x	p	v	a)v>0 Nota encendida p=altura v=velocidad	1,2
			b)v=0 Nota apagada p=altura	1,2
$Ax	p	pr	Polifónico después de pulso p=altura pr=presión	1,3a
$Bx	n	c	a)n<$7A Cambio de control n=número controlador c=valor controlador	4
			b)n=$7A Control local on(c=$7F) off(c=0)	5
			c)n=$7B,c=0 Todas las notas apagadas	6
			d)n=$7C,c=0 Mod. omni apagada (todas las notas apagadas)	6
			e)n=$7D,c=0 Mod. omni encendida (todas las notas apagadas)	6
			f)n=$7E Mod. mono encendida (todas las notas apagadas) c= número de canales	6,7
			g)n=$7F,c=0 Mod. poli encendida (todas las notas apagadas)	6
$Cx	pp	—	Selección Prog. Parche pp=número de programa	
$Dx	pr	—	Presión de canal pr=valor de presión	3b
$Ex	l	m	Cambio mando de altura l=7 LSBs m=7 MSBs	8

Notas:
Excepto donde se especifique otra cosa, todos los bytes de datos pueden tener cualquier valor entre 0 y $7F. El número de canal se representa mediante x, el dígito (hexa) menos significativo del byte de estado (x=0 especifica canal 1 y x=$F especifica canal 16).
1. p es el tono de la nota en semitonos. *Do central* es el número de tono $3C y, por tanto, todos los *do* son múltiplos de $0C (decimal 12). El teclado de piano estándar de 88 notas va de $15 a $6C.
2. Los valores de velocidad van desde $01 hasta $7F. Los teclados sin sensores de velocidad deben utilizar una velocidad por defecto de $40. Un mensaje de «nota encendida» con una velocidad de 0 equivale a un mensaje de «nota apagada» con una velocidad por defecto de $40.
3. «Después de pulsación» es la cantidad de presión ejercida sobre una tecla tras haberla pulsado. Por lo general se utiliza para introducir efectos de modulación sin tener que operar un mando de modulación. Hay dos tipos de sensores de presión y dos tipos diferentes de mensajes.
a) El "después de pulsación" individual o polifónico requiere sensores de presión individuales para cada tecla y afecta sólo a la nota correspondiente a la tecla pulsada. En consecuencia, además del valor de presión se debe enviar un número de tono. (Su implementación requiere, asimismo, que se proporcionen circuitos de modulación separados para cada voz.)
b) La presión de canal se produce mediante un único sensor de presión, al que afectan de igual forma todas las teclas. La presión que se transmite corresponde a la presión máxima (todas las teclas pulsadas al mismo tiempo). El efecto resultante se aplica simultáneamente a todas las voces.
4. Estos mensajes los envían controladores separados del teclado. Los números controladores de $0 a $1F son controladores continuos que poseen valores de control (c) de $0 a $7F. Éstos equivalen a las palancas de mando de potenciómetro para ordenadores y de hecho pueden ser palancas de mando u otros dispositivos tales como mandos de ruedas, pedales y controladores de aliento. Los números de controlador de $20 a $3F se utilizan opcionalmente para enviar siete bits menos significativos extras para los controladores de $0 a $1F, si se requiere una resolución muy alta. Los números de controlador del $40 al $5F son controladores de conmutación (como los pedales de sostenimiento, portamento, interruptores on/off, etc.) con c establecido ya sea en cero (off) o bien en $7F (on). Éstos son equivalentes al tipo de palanca de mando de conmutación más usual que utilizan los ordenadores. Los números de controlador del $60 al $79 son indefinidos, y los del $7A al $7F están reservados para mensajes de modalidad de canal. No se requiere que se asignen los números de controlador a controladores físicos específicos, con la excepción del mando de modulación, al que se suele asignar el $1.
5. El control local se utiliza opcionalmente para romper la unión interna entre los dispositivos de entrada de una unidad (por lo general, un teclado y controladores asociados) y su sistema de circuitos de generación de sonido, de modo que el teclado (p. ej.) sólo envía datos a MIDI OUT y los circuitos de generación de sonido sólo responden a los datos recibidos en MIDI IN.
6. La implementación de «todas las notas apagadas» es opcional, y estos mensajes no se deben utilizar en lugar de instrucciones individuales de «nota apagada» para apagar varias notas. El razonamiento que motiva esto no está nada claro y hace que las instrucciones sean virtualmente inútiles.
7. El tercer byte del mensaje de modalidad mono especifica el número total de canales requerido. Si este número es cero, la cantidad de canales es igual a la cantidad de voces disponibles en el receptor.
8. El mando de altura se diferencia de otros controladores en que puede tener valores positivos y negativos. La posición central se envía como l=$0 y m=$40. Muchos receptores poseen sólo siete bits de resolución y sólo responderán a m, ignorando el valor de l

Mensajes del sistema

Estado	Datos1	Datos2	Descripción	Notas
$F0	cualquier número		Exclusivo del sistema	1
$F1			Indefinido	
$F2	l	m	Señalador posición canción l=7 LSBs m=7 MSBs	2
$F3	s		Selección de canción s=número de canción	
$F4			Indefinido	
$F5			Indefinido	
$F6			Solicitud de melodía	3
$F7			Fin de exclusivo del sistema	1
$F8	—	—	Reloj temporizador	4
$F9			Indefinido	
$FA	—	—	Empezar	4
$FB	—	—	Continuar	4
$FC	—	—	Parar	4
$FD			Indefinido	
$FE	—	—	Percepción activa	5
$FF	—	—	Inicialización del sistema	6

Notas:
Los mensajes de $F8 a $FF son los mensajes del sistema en tiempo real y se pueden enviar en cualquier momento (incluso durante la transmisión de otros mensajes).

1. Los mensajes exclusivos del sistema pueden tener cualquier número de bytes de datos, terminando con un byte de «fin del exclusivo del sistema» ($F7) o cualquier otro byte de estado.

2. Este mensaje se utiliza para preestablecer arbitrariamente el puntero de posición de canción, que es un registro interno que contiene la cantidad de *beats* (1 *beat*=6 relojes MIDI) desde el comienzo de la secuencia (canción).

3. Éste se utiliza para solicitar a los sintetizadores analógicos que afinen sus osciladores.

4. Estos mensajes se emplean para sincronizar las unidades secuenciadoras maestra y esclava. El reloj del sistema se establece a una velocidad de 1/24avo de un cuarto de nota. La instrucción «continuar» se diferencia de «comenzar» en que reinicia una secuencia desde el puntero en el cual se detuvo.

5. La percepción activa se utiliza como un mensaje ficticio cuando en la MIDI no se registra ninguna otra actividad. Si se utiliza, debe ser enviado de modo que no transcurran más de 300 ms sin actividad.

6. Este mensaje se emplea para inicializar el sistema completo a la situación de encendido. No se lo debe enviar automáticamente detrás del encendido, para impedir la posibilidad de que dos unidades se reinicialicen la una a la otra indefinidamente.

No se deben enviar mensajes que no estén incluidos en la lista anterior, y los receptores habrán de ignorarlos

Pequeño melocotón

Si bien el Apricot F1e es principalmente una máquina de gestión, puede también suscitar interés en el campo educativo

Chris Stevens

Aspecto atrayente
El Apricot F1e ha heredado la apariencia atractiva de sus parientes. La unidad de disco integral de 3 ½ pulgadas, el sistema operativo MS-DOS estándar y los paquetes de software en lote, junto con un precio asequible, hacen del F1e un serio competidor en los círculos educativos y también entre los usuarios serios de ordenadores personales

Quienes estén familiarizados con los ordenadores Apricot (albaricoque) reconocerán fácilmente la estructura básica del F1e. Vendido en forma de paquete, incluye el teclado F1, la unidad del ordenador propiamente dicha con una unidad de disco de una sola cara y densidad simple, y una pantalla de fósforo verde de ocho pulgadas. Al igual que con el Apricot Portable, la empresa ha aplicado su tecnología de infrarrojos, que elimina la necesidad de usar gran número de cables colgantes. Tanto el teclado como el ratón opcional controlan el ordenador a través de haces infrarrojos que son detectados por sensores en la parte delantera del F1e. Haciéndose eco de la tendencia hacia ordenadores cada vez más pequeños (iniciada con el Apple Macintosh), que no ocupan excesivo espacio, el F1e es notablemente estrecho, midiendo apenas 200 mm de ancho, aunque esto queda compensado por su longitud, que es de alrededor de 425 mm.

El teclado tiene el trazado Apricot estándar; las teclas están niveladas y su aspecto es similar a las del Sinclair QL. No es particularmente apropiado para mecanografía al tacto y, en consecuencia, no se presta muy bien a aplicaciones de tratamiento de textos. Las teclas, si bien son adecuadas para una máquina de gestión moderna, traquetean un poco y pueden despertar dudas en cuanto a su fiabilidad a largo plazo. Al igual que en el Apricot Portable, en la parte superior del teclado hay cuatro botones: «Reset», para arranques en frío, «Repeat Rate», que permite variar la velocidad a la cual se repiten los caracteres cuando se mantiene pulsada una tecla, «Set Time» y «Keyboard Lock».

La unidad del ordenador posee una única unidad de disco Sony de 3 ½ pulgadas, incorporada en la parte anterior, que está adquiriendo una creciente popularidad entre los fabricantes y en consecuencia ha justificado el uso original de ACT de este formato para sus máquinas en más de un sentido. Mientras que otras empresas están teniendo dificultades para transferir sus amplias bases de software a los discos de 3 ½ pulgadas, todo el software Apricot es convenientemente compatible. En consecuencia, los usuarios que deseen adquirir el F1e no tienen necesidad de preocuparse por la falta de software adecuado. Además, los discos de una sola cara y densidad simple que utiliza el F1e pueden retener un máximo de 315 K de información, más que sus equivalentes de 5 ¼ pulgadas.

Otras características de la parte delantera de la unidad de ordenador incluyen una serie de LEDs que indican potencia, Caps Lock, desplazamiento de la pantalla y unidad de disco *on/off*. Debajo de ellos están los detectores de haces infrarrojos.

En el lado derecho del ordenador hay un bus de ampliación de 60 vías, a través del cual se puede añadir una amplia gama de placas de ampliación o una unidad de disco extra. Esta ranura hace que el F1e se pueda ampliar a la misma capacidad que el Apricot XL mediante la adición del sistema de ampliación MSD, que proporciona 10 megabytes en disco rígido.

El panel posterior, si bien no posee muchas interfaces, está diseñado suficientemente bien como para permitir la instalación de los periféricos más comunes. Sobre la izquierda hay un conector D de 25 vías estándar que da cabida a una interface en serie RS232. Junto a esta puerta hay dos conectores para pantalla. El primero es un adaptador de nueve patillas que proporciona la señal RGB para pantallas en color (si bien las pantallas ACT monocromáticas existentes también se enchufan en este conector). A la derecha de éste hay un conector de video compuesto para otros tipos de pantallas.

El precio del F1e no incluye la pantalla. El ordenador no posee ningún adaptador RF que lo capacite para enviar una señal a un aparato de televisión común; no obstante, sí hay disponible un adaptador

que proporciona tal señal. Asimismo, dado que el F1e estándar no tiene pantalla instalada y que muchas pantallas poseen su propia entrada de potencia, los conectores para pantalla no proporcionan líneas de potencia de forma automática, y el ordenador propiamente dicho no es capaz de proporcionar tal fuente. Por lo tanto, para que éste pueda operar una de las pantallas ACT, se le ha de instalar una fuente de alimentación externa de 17 V.

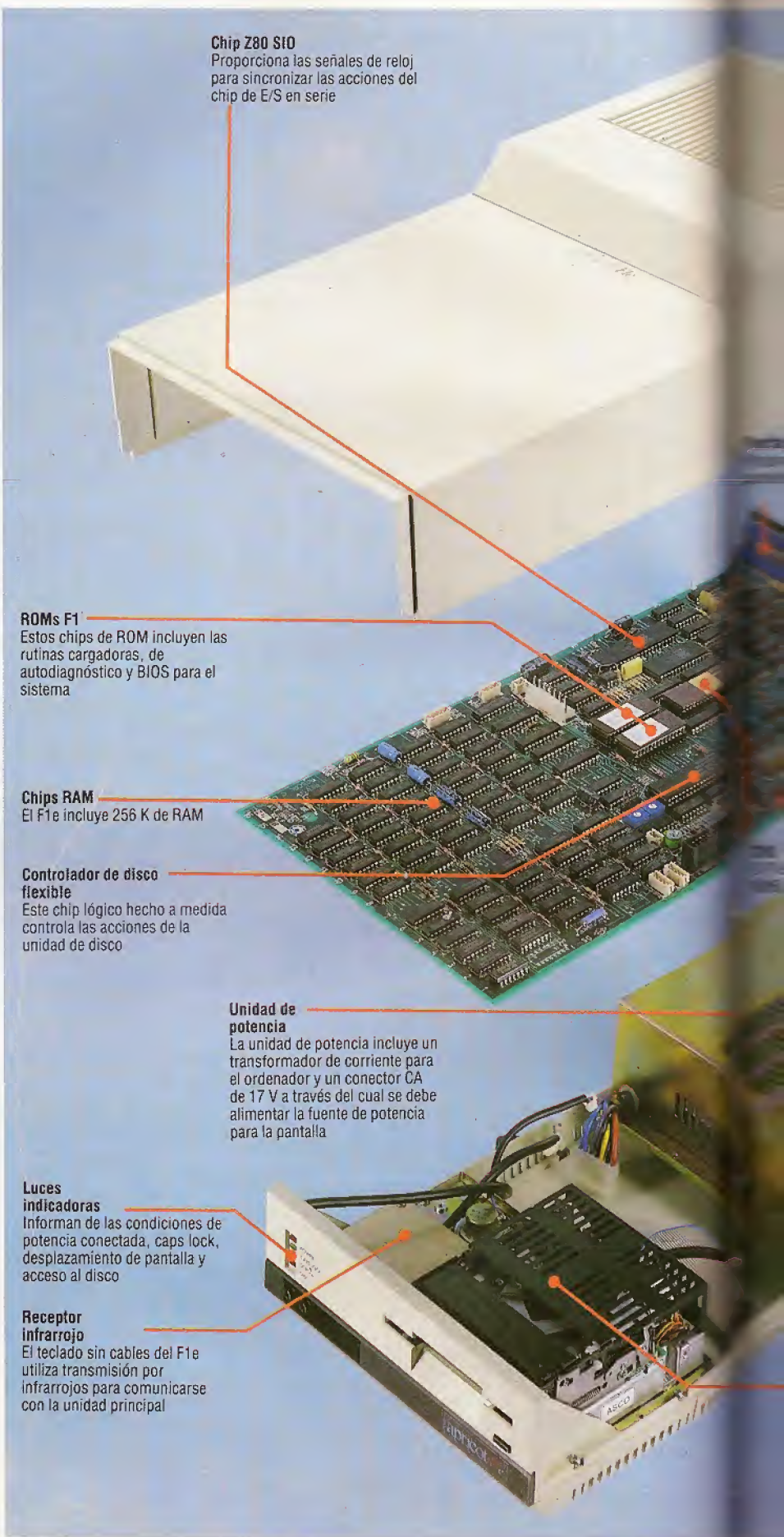
Aunque esto parezca lógico, el hecho de que ACT haya tomado la decisión de suprimir los cables colgantes mediante la incorporación de comunicaciones infrarrojas, no concuerda con su decisión de proporcionar una fuente de alimentación externa para operar sus pantallas. Esto adquiere especial relevancia teniendo en cuenta que otros modelos de la gama (como el ACT Apricot, el buque insignia de la empresa) no requieren tal dispositivo. La fuente de alimentación se puede colocar en un lugar donde no moleste, en especial cuando se la instale en una posición permanente, pero así y todo uno queda con la sensación de que, en este aspecto, la máquina no se pensó lo suficiente.

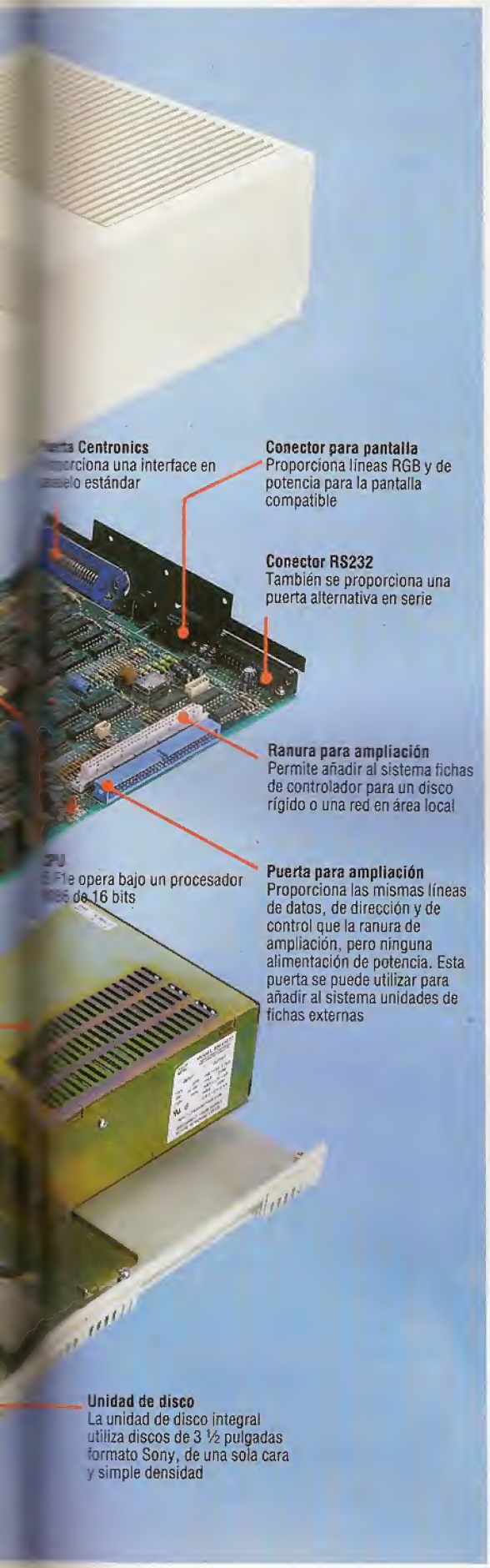
En el interior del ordenador, hay otra ranura de ampliación para placas adicionales. La placa del F1e, al igual que otras de la gama Apricot, tiene un diseño elegante y está protegida de la unidad de disco y el transformador de potencia (que puede generar temperaturas perjudiciales) mediante una carcasa de metal que actúa a modo de disipador.

El F1e está basado en el procesador 8086 de 16 bits y opera bajo el popular sistema operativo MS-DOS. Los ordenadores Apricot, sin embargo, siempre han tenido su propio sistema de administración basado en iconos, lo que en realidad significa que su relación directa con el sistema operativo es escasa. El sistema de administración del F1e se llama Activity y es con él con quien los usuarios se familiarizarán más.

Activity es un programa orientado hacia objetos. Ello significa que, en lugar de impartir una serie de instrucciones para el ordenador, como una para cargar (LOAD) un archivo en la memoria, uno sola-

Software de calidad
El software gratuito que se entrega con el F1e incluye un procesador de textos, un cuaderno de notas electrónico y una hoja electrónica. Aunque los dos primeros programas son, relativamente, poco sofisticados, la hoja electrónica (*SuperCalc*) posee muchas facilidades, como la división en ventanas, que cabe esperar de todas las aplicaciones caras de este tipo





Chris Stevens

mente indica (por lo general seleccionando un icono entre una serie presentada en la pantalla) la acción que desea realizar: el ordenador se encargará del resto.

La mayoría de estos sistemas dependen del uso de un ratón para desplazar el cursor a través de la pantalla; pero en el sistema F1e el mismo se puede sustituir por el teclado numérico. Los números del 1 al 9 están dispuestos en tres filas de tres teclas. Tomando el teclado a modo de brújula (excluyendo la tecla central) hay ocho direcciones en las cuales se puede mover el cursor. Por lo tanto, pulsando la tecla central de la fila superior (8) el cursor se moverá directamente hacia arriba hasta un icono que se halle en esa dirección, mientras que al pulsar 3 (tecla de la derecha de la fila inferior) el cursor se moverá oblicuamente hacia abajo y hacia la derecha. Para seleccionar un icono se debe pulsar la tecla Enter del teclado de calculadora.

Entre el juego de programas incluido en el disco del sistema Activity hay un programa de aprendizaje para ayudar al usuario a acostumbrarse al sistema. Asimismo, explica los usos del icono y los editores de fuentes (con facilidades para cargar sus propios juegos de caracteres cuando así lo quiera) y el configurador del sistema, que permite preparar el ordenador para cualquier periférico especializado que pueda estar utilizando.

Como cabe esperar en una máquina pensada fundamentalmente como máquina de gestión, el F1e incluye varios paquetes de software en un lote de aplicaciones. *SuperWriter* es un programa para tratamiento de textos basado en el *WordStar*, cuyo uso está tan difundido, pero sin las amplias capacidades de formateo de este último. El *SuperPlanner* se describe como un «bloc de notas electrónico» que permite al usuario «planificar con antelación» sus actividades. Este programa incluye una agenda de direcciones, un calendario, un diario y un pequeño sistema de archivo. Aunque superficialmente se asemeja a una base de datos, *SuperPlanner* no contiene las refinadas técnicas de búsqueda y recuperación que por lo general caracterizan a una auténtica base de datos.

El último paquete que viene empaquetado con el F1e es *SuperCalc*. Éste es una hoja electrónica para aplicaciones financieras y contables. Quizá sea el más completo de los tres paquetes; contiene una gran cantidad de instrucciones que permiten ventanas, justificación de textos y muchas otras facilidades que es dable esperar de un paquete de hoja electrónica profesional.

Al reducir el precio del F1e, ACT ha dado a entender que tiene intenciones de introducirse de forma concertada en el mercado educativo. Esta actitud se hace aún más evidente al haber lanzado ACT, paralelamente al anunciado recorte de precios, un nuevo producto denominado B-TRAN, que permite que el ordenador ejecute virtualmente todos los programas escritos en BASIC BBC.

Ciertamente, la reducción del precio del F1e mejorará las ventas del ordenador, en particular en los mercados educativo y de pequeña gestión. Si bien no es probable que llegue a dominar el mercado del ordenador personal, el F1e es, no obstante, una compra excelente para quienes estén interesados en adquirir una máquina de precio asequible que se puede ampliar hasta alcanzar la potencia de un ordenador de gestión totalmente equipado.

APRICOT F1e

DIMENSIONES
425 × 200 × 105 mm

CPU
Procesador Intel 8086 operando a 4.7 MHz

MEMORIA
256 Kbytes estándares

PANTALLA
Resolución de textos de 132 × 50 u 80 × 25 caracteres, o una resolución máxima para gráficos de 800 × 400 pixels

INTERFACES
Puerta RS232, interface Centronics y conectores para RGB y video compuesto

UNIDAD DE DISCO
Una sola unidad de disco de 3 ½ pulgadas y 315 Kbytes

SISTEMA OPERATIVO
MS-DOS, CP/M-86 y Concurrent CP/M

DOCUMENTACION
El manual del paquete de aplicaciones es muy completo y responde al elevado estándar que es habitual en ACT, si bien el manual de iniciación es algo parco en cuanto a información detallada sobre la máquina en sí misma

VENTAJAS
Por su precio, el F1e parece una ganga y podría causar un gran impacto en los mercados educativo y de pequeña gestión

DESVENTAJAS
El teclado plano no está a la altura del estándar del ACT Apricot. Aunque contiene un procesador rápido, el F1e no es tan veloz como algunas máquinas que pueden compararse con él

Datos básicos (I)

Por cortesía de la Commodore Business Machines, iniciamos un análisis detallado del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
D6510	0000	0	Chip 6510 On-Registro de dirección de datos
R6510	0001	1	Chip 6510 On-reg. E/S 8 bits
	0002	2	No usado
ADRAY1	0003-0004	3-4	Vector saltos: conversión flotante-entero
ADRAY2	0005-0006	5-6	Vector saltos: conversión flotante-entero
CHARAC	0007	7	Carácter búsqueda
ENDCHR	0008	8	Flag: rastrea comillas de fin de variable en serie
TRMPOS	0009	9	Col. pantalla desde últ. TAB
VERCK	000A	10	Flag: 0=carga, 1=verificar
COUNT	000B	11	Puntero buffer entrada / n.° subscritos
DIMFLG	000C	12	Flag: DIMensión tabla por defecto
VALTYP	000D	13	Tipo datos : $FF=serie,$00=número
INTFLG	000E	14	Tipo datos : $80=entero,$00=flotante
GARBFL	000F	15	Flag: busca DATA/comillas LIST/recolector de basura
SUBFLG	0010	16	Flag: refer. suscrito/llamada función usuario
INPFLG	0011	17	Flag:$00=INPUT,$40=GET, $98=READ
TANSGN	0012	18	Flag: signo TAN/Resultado comparación
	0013	19	Flag: interrogación de INPUT
LINNUM	0014-0015	20-21	Temp: valor entero
TEMPPT	0016	22	Puntero: pila temporal de serie

Marcus Wilson-Smith

Movimiento de apertura
Tradicionalmente, el juego del *go* se desarrolla sobre un tablero de madera que tiene grabada una cuadrícula de 19 por 19 líneas que se intersectan. Para nuestra versión informatizada hemos reducido el tamaño del tablero a una cuadrícula de 15 por 15, para permitir que se pueda visualizar cómodamente en una pantalla

Jugadas preliminares

Cuando se escribe un programa para un juego como el «go», es mejor empezar por las rutinas de E/S

El programa de *go* se presentará en cuatro versiones separadas: para los micros BBC Modelo B, Commodore 64, Sinclair Spectrum y Amstrad CPC 464/664. Siempre que ha sido posible, los números de línea se han mantenido constantes para todas las versiones, pero por diversas razones ha sido necesario incluir líneas extras para implementar algunas rutinas en cada una de las máquinas. Esto es especialmente evidente en el caso de la recursión empleada en el BASIC BBC (no está implementada la recursión en ninguna de las otras tres máquinas). Por consiguiente, es necesario implementar una pila para el usuario con el fin de conseguir que los listados sean lo más parecidos posible. Cuando el texto hace mención a rutinas y variables del listado, éstas corresponden a la versión para el BBC Micro, si bien las correspondencias entre los cuatro listados son bastante directas.

La siguiente sección del programa se ocupa de la inicialización de variables, la creación de la visualización del tablero y las rutinas de entrada. Las líneas de la 10 a la 140 constituyen el cuerpo principal del programa. Antes de entrar el bucle del juego propiamente dicho (de la línea 60 a la 90), se llamará a las rutinas PROCinicializar y PROCintroduccion.

PROCinicializar sólo se utiliza en la primera ejecución del programa, para DIMensionar el tablero (tablero%), inicializar el cursor, etc. El tablero se DIMensiona para estar compuesto por una serie de 255 bytes, creando una superficie de juego de 15 por 15 (en lugar del tablero normal del *go*, de 19 por 19). Esto obedece a varias razones, de las cuales la más importante es la velocidad. No se puede esperar que un programa de juego en BASIC opere con especial rapidez, pero reduciendo las dimensiones del tablero el tiempo de ejecución se acortará en aproximadamente un tercio, sin incidir en la forma de jugar al juego. Por supuesto, esto también significa que el tablero cabrá limpiamente en una página (256 bytes, un cuarto de byte) de memoria, permitiendo un límite compuesto por cuadrados individuales. Asegurando que todas las referencias al tablero estén comprendidas en la escala entre 0 y 255, este reborde circundará el tablero.

Las variables inicializadas en las líneas 190 y 200 son constantes, pero el uso del nombre de la variable en lugar de números simplifica mucho las modificaciones. Por ejemplo, si usted quisiera un juego para dos personas, o bien que el ordenador jugara consigo mismo, sólo necesitaría cambiar los valores de blancas% y negras% para poder utilizar normalmente todas las rutinas. Sobre la base de estos valores, cada byte del tablero se empleará de la forma que se indica en el diagrama.

PROCintroduccion se utiliza al inicio de cada partida, para preparar el contador de movimientos (movimientos%), la condición de final (fin%), etc. Hace uso de las rutinas PROCinic_juego y PROCpantalla_titulos. La pantalla de títulos es bastante elemental debido a las restricciones de espacio para los listados, pero usted está en libertad de modificarla en la medida que así lo desee. Por ejemplo, podría añadir el símbolo japonés del *go* o, quizá, algunos acordes de música oriental. Si desea añadir trozos de código, puede emplear para ello todos los números de línea a partir del 5000.

PROCimprimir_tablero, como es natural, imprime el tablero. FNint_to_char se utiliza desde esta rutina para convertir una coordenada de tablero en enteros en las coordenadas de caracteres que emplea la visualización en pantalla; se DEFine en la línea 2260.

FNinput y PROCmensaje son dos rutinas de propósito general para la entrada del usuario y la impresión de mensajes, basadas en la matriz mens$ inicializada en PROCleer_mensajes. Esta matriz retiene 10 avisos o mensajes que el ordenador necesita visualizar durante el juego. El mensaje apropiado se

selecciona pasando a estas dos rutinas el número del elemento de la matriz correspondiente, M%. La incorporación de estas rutinas, en lugar de limitarse a solicitar y recibir una entrada y una salida cuando es necesario, aumenta la flexibilidad del programa.

Al digitar la versión para el Spectrum, puede encontrarse con un problema entre las líneas 1480 y 1505. Los números de estas líneas subrayados hacen referencia a los gráficos de símbolos del Spectrum. Usted los obtendrá pulsando primero Caps Shift y 9 juntas, entrando por tanto en modalidad de gráficos (el cursor G) y después digitando los números indicados. Pulse nuevamente Caps Shift y 9 para salir de la modalidad de gráficos al final de la sentencia de impresión. Si el número va precedido por sh, entonces digite el número mientras pulsa la tecla Shift al mismo tiempo.

Si ejecuta el código tal como está, obtendrá una pantalla de títulos pidiéndole una cantidad de fichas de handicap, seguida por la pantalla principal del juego. Luego el programa hará indefinidamente un bucle. Por el momento el número de handicap se ignora, pero en el próximo capítulo lo añadiremos, así como algunas otras rutinas generales.

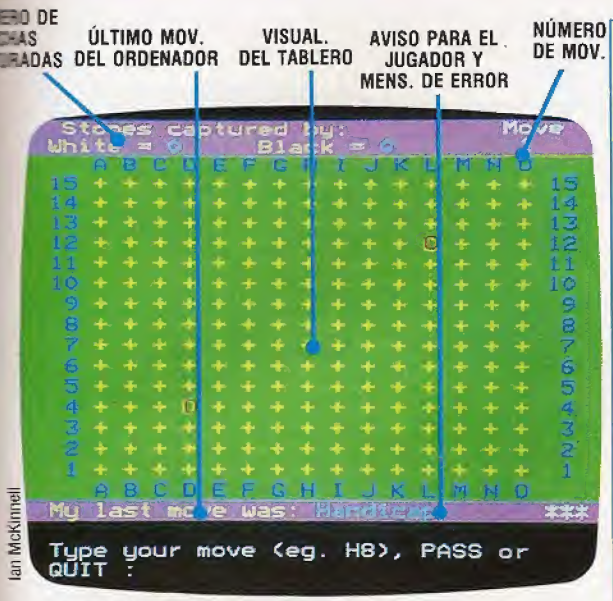
Variables del juego del «go»

Variable	Finalidad
negras%	Valor 1: ficha negra en byte del tablero.
tablero%	El valor de comienzo para el tablero de 256 bytes en la memoria.
captura%(2)	Retiene la cantidad de fichas capturadas por las negras y las blancas.
color%	Valor 3. Se utiliza para enmascarar los bits de color en un byte del tablero.
dir%(4)	Retiene los desplazamientos necesarios para avanzar un cuadrado hacia el N, E, S y O.
licencia%	Valor 8. Se usa en la rutina de búsqueda de grupos para marcar licencias ya contadas.
marcador%	Valor 4. Se usa en las rutinas de búsqueda de grupos para marcar fichas ya contadas.
movimiento%	Retiene la cantidad de movimientos efectuados en cada partida para la visualización.
blancas%	Valor 2: ficha blanca en byte del tablero.
atari1$	Retiene 5 espacios cualesquiera o «Atari» si el último mov. del ord. produjo esta sit.
atari2$	Retiene 5 espacios cualesquiera o «Atari» si el último mov. del jugador produjo esta situación. Nota: Atari es la acción de colocar una ficha de modo tal que deje a uno o más grupos del oponente con una sola licencia.
eje$	Parte de la visualización del tablero.
mens$	Retiene todos los mensajes generales de E/S, utilizados por PROCinput y PROCmensaje.

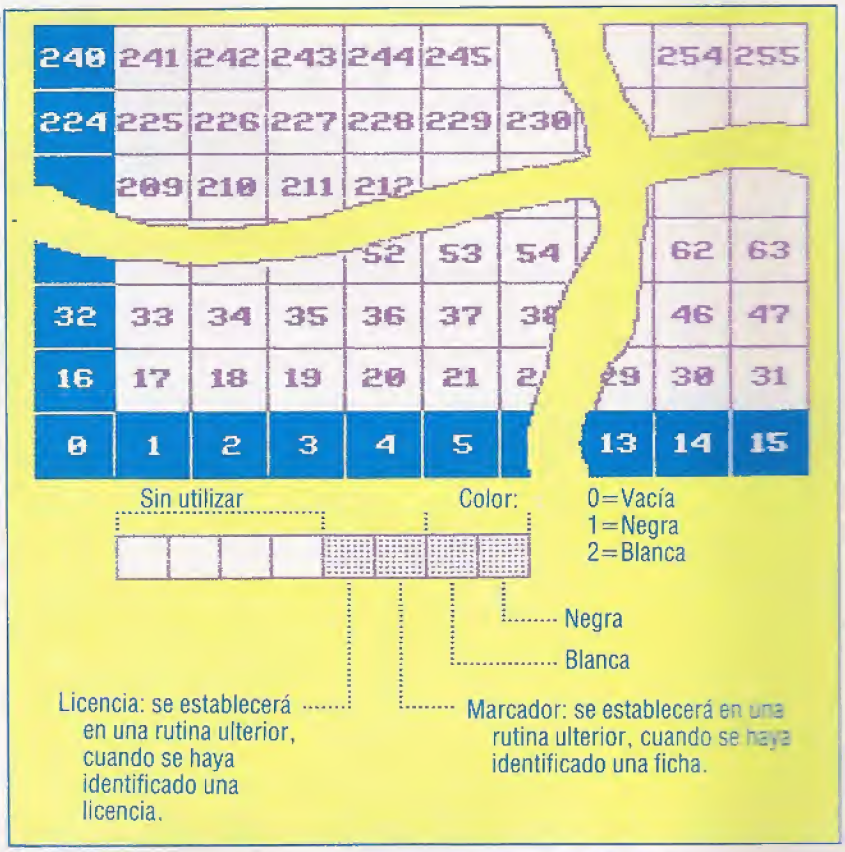
Módulo Uno

BBC Micro:

```
 10 MODE 7
 20 *FX14,6
 30 PROCinicializar
 40 PROCintroduccion
 50 CLS: PROCimprimir_tablero
 60 movimiento%=movimiento%+1: REM aqui movimientos de las
    blancas
 70 IF fin% GOTO 100
 80 movimiento%=movimiento%+:REM aqui movimientos de las
    negras
 90 IF NOT fin% GOTO 60
100 resp$=FNinput (21,9,1)
110 IF resp$="S» GOTO 40 ELSE IF resp$<>«N» GOTO 100
120 PROCmensaje (22,5,"")
130 PRINT: END
140 :
150 REM ********************************************
160 :
170 DEF PROCinicializar
180 LOCAL L%
190 negras%=1 : blancas%=2 : color%=3
200 marcador%=4 : licencia%=8
210 @%=2
220 DIM tablero% 255
230 VDU 23;8202;0;0;0;
240 DIM captura%(2)
250 eje$=CHR$130+CHR$157+CHR$132+"A B C D E F G H I J K
    L M N O"
260 PROCleer_mensajes
290 DIM dir%(4)
300 RESTORE 340
310 FOR L%=1 TO 4
320     READ dir% (L%)
330     NEXT
340 DATA 16,1,-16,-1
350 ENDPROC
360 :
370 REM ********************************************
380 :
390 DEF PROCleer_mensajes
400 LOCAL M%
410 RESTORE 460
420 DIM mens$(9)
430 FOR M%=0 TO 9
440     READ mens$(M%)
450     NEXT
460 DATA "O.K. ESTOY PENSANDO..."
470 DATA "Entrada ilegal: "
480 DATA "Ficha ya en ese lugar: "
490 DATA "Ilegal. Ko en ese lugar: "
500 DATA "Ilegal. Suicidio en ese lugar: "
510 DATA "    O.K. JUEGO TERMINADO."
520 DATA ""
530 DATA "    Cuantas fichas puedo tener de handicap
    (2-9) ?"
540 DATA "Digite su movimiento (por ej. H8), PASO o
    ABANDONO :"
550 DATA "Quieres jugar otra partida (S/N)?"
560 ENDPROC
570 :
580 REM ********************************************
1260 :
1270 DEF PROCintroduccion
1280 PROCinic_juego
1290 PROCpantalla_titulos
1300 ENDPROC
1310 :
1320 REM ********************************************
1330 :
1340 DEF PROCinic_juego
1360 atari1$="     ":atari2$="     "
1370 posicion%=0:movimiento%=1
1380 fin%=FALSE
1390 captura%(1)=0:captura%(2)=0
1410 ENDPROC
1420 :
1430 REM ********************************************
1440 :
1450 DEF PROCpantalla_titulos
1460 CLS
1470 PRINT TAB(12,4);CHR$145;CHR$154;"h7+$
     h7k4"
1480 PRINT TAB(12,5);CHR$145;CHR$154;"j5k5
     j5j5"
1490 PRINT TAB(12,6);CHR$145;CHR$154;"pssp
     pssp"
```



Comentando la partida
La visualización en pantalla de arriba muestra la disposición de las diversas características del juego. Además de mostrar el tablero, el programa informa al jugador sobre el número de fichas capturadas tanto por el ordenador como por el jugador, el número de movimientos realizados en la partida, y cuáles fueron los últimos movimientos efectuados por la máquina y por su adversario. La porción inferior de la pantalla está reservada a avisos y mensajes de error dirigidos al jugador. A medida que se desarrolle el juego veremos cómo el programa detecta y desautoriza movimientos ilegales como el «suicidio» o cualquier intento por colocar fichas en posiciones ya ocupadas




```
1500 PRINT TAB(9,10);CHR$134;"por Marcus
     Jeffery"
1510 PRINT TAB(1,13);CHR$133;«Jugaras con
     las";
1520 PRINT CHR$135;"fichas";CHR$133;"blancas,
     y"
1530 PRINT CHR$133;"el ordenador (al ser mas
     débil!)"
1540 PRINT CHR$133;"jugara con las";CHR$129;"fichas";
     CHR$133;"negras con ventaja de"
1550 PRINT TAB(13);CHR$133;"handicap"
1560 hand%=VAL(FNinput (20,7,1))
1570 IF hand%<2 OR hand%>9 THEN 1560
1590 ENDPROC
1600 :
1610 REM ***********************************************
1720 :
1730 DEF PROCimprimir_tablero
1740 LOCAL P%, X%, Y%
1750 PRINT TAB(0,0);CHR$133;CHR157;CHR$131;"Fichas
     capturadas por:";
1760 PRINT TAB(32,0);CHR$135;"movimiento"
1770 PRINT TAB(0,1);CHR$133;CHR$157;CHR$131;"Blancas
     =";CHR$134;captura%(2);
1780 PRINT TAB(16,1);CHR$131;"Negras
     =";CHR$134;captura%(1);
1790 PRINT TAB(33,1);CHR$135;movimiento%
1800 PRINT TAB(0,2); eje$
1810 FOR Y%=15 TO 1 STEP -1
1820    PRINT TAB (0,18-Y%);CHR$130;CHR$157;CHR$132;
     RIGHT$("  "+STR$ (Y%),2);
1830    FOR X%=1 TO 15
1840    P%=tablero%?(16*Y%+X%)
1850    IF P%=1 THEN PRINT CHR$129;"O";:GOTO
        1880
1860    IF P%=2 THEN PRINT CHR$135;"O";:GOTO
        1880
1870    PRINT CHR$131;"+";
1880    NEXT
1890    PRINT TAB(35,18-Y%);CHR132,Y%
1900    NEXT
1910 PRINT TAB(0,18);eje$
1920 PRINT TAB(0,19);CHR$133;CHR$157;CHR$131;"Mi ultimo
     movimiento fue:";
1930 PRINT CHR$134;FNint_to_char(posición%);TAB(30)
     atari1$
1940 PRINT TAB(16,22);atari2$;CHR$7
1950 ENDPROC
1960 :
1970 REM ***********************************************
1980 :
1990 DEF FNinput(P%,M%,W%)
2000 LOCAL S%,A$,I$
2010 PRINT TAB(39,P%+1);STRING$(119,CHR$127);
2020 A$=""
2030 PRINT TAB(3,P%);mens$(M%);CHR$134;
2040 PRINT STRING$(W%,CHR$32);STRING$(W%,
     CHR$127);
2050 *FX21,0
2060 I$=GET$ : IF ASC(I$)=13 THEN GOTO 2120
2070 IF ASC(I$)<>127 GOTO 2090
2080 IF S%>0 THEN S%=S%-1:A$=LEFT$(A$,S%):GOTO 2110
     ELSE GOTO 2060
2090 IF I$>="a" AND I$<="z" THEN I$=CHR$(ASC(I$)
     -32)
2100 IF S%<W% THEN S%=S%+1:A$=A$+I$ ELSE GOTO
     2060
2110 PRINT I$; : GOTO 2060
2120 =A$
2130 :
2140 REM ***********************************************
2150 :
2160 DEF PROCmensaje(P%,M%,A$)
2170 LOCAL L%
2180 FOR L%=P% TO P%+ 1
2190    PRINT TAB(39,L%);STRING$(39,CHR$127);
2200    PRINT CHR$141;CHR$136;CHR$129;mens$(M%);
        A$;
2210    NEXT
2220 ENDPROC
2230 :
2240 REM ***********************************************
2250 :
2260 DEF FNint_to_char (P%)
2270 IF P%=0 THEN ="Handicap"
2280 =CHR$(P% MOD 16 + 64)+STR$(P% DIV 16)+" "
2290 :
2300 REM ***********************************************
```

Bits a bordo
Cada posición del tablero se representa en la memoria mediante un único byte. En el diagrama superior vemos el trazado de los 255 bytes que representan todas las intersecciones en el tablero de 15 por 15. Los bits de cada byte retienen información sobre el estado de la intersección correspondiente del tablero. Los dos bits menos significativos determinan el color de la ficha presente (o indican si la intersección está libre). Los bits 2 y 3 los utilizarán las rutinas de evaluación del «estado del juego», que desarrollaremos en próximos capítulos

Plan de acción

Finalmente veremos cómo actúan dos importantes paquetes: «Sycero» y «The Last One»

Todo el software destinado a posibilitar que el principiante absoluto genere aplicaciones para ordenador debe satisfacer algunos puntos rigurosos:

- Debe favorecer un enfoque *de arriba abajo,* en el que el usuario pueda definir el proyecto a realizar y especificar con claridad cada punto a medida que vaya progresando el trabajo.
- Debe ser activado por menú, con solicitudes de opciones múltiples que vayan guiando al usuario a través de las opciones disponibles.
- Debe detectar los errores apenas se produzcan y permitir cambios o correcciones simples.
- El código resultante debe ser «transportable», es decir, debe poder ser ejecutado en otro ordenador, con independencia del software del sistema que lo generó.
- Debe producir amplia documentación del proceso de generación del programa, de modo que el usuario que no esté familiarizado con la programación precedente pueda introducir correcciones o mejoras.

Sin embargo, puesto que el generador de programas es una herramienta cuyo uso no está limitado a los no iniciados, también ha de resultar aceptable al usuario más experimentado. Es probable que un programa dirigido a un nivel muy bajo moleste e incluso confunda al usuario más avanzado, además de demorar las cosas con medidas de protección y seguridad innecesarias. También es probable que el usuario más experimentado tenga algunas rutinas favoritas (formas de usar el BASIC, p. ej.) y el generador habrá de estar preparado para aceptarlas. Teniendo en cuenta todas estas consideraciones, los dos generadores de programas de mayor aceptación, *The Last One (TLO)* y el más reciente *Sycero,* obtienen altas calificaciones.

Ambos utilizan ampliamente menús informativos. *Sycero* posee pantallas de ayuda para los controles del cursor y gráficos; las instrucciones para edición de textos están disponibles en todo momento pulsando las teclas Control y H. De los dos sistemas, el *TLO* es el que sigue con más fidelidad el enfoque de arriba abajo. Comienza por elaborar un diagrama de flujo, que en realidad es una serie de comentarios (REM) acerca de lo que se pretende que haga cada sección del programa (Branch on 4-option menu, p. ej.) y, si así se desea, se lo puede incorporar al final del programa en BASIC.

Las pantallas de entrada y de visualización se definen durante el proceso de codificación del programa y el código resultante estará en formato ASCII. El usuario entonces ha de salir del generador, cargar (LOAD) el programa generado y volver a guardarlo (SAVE) en código binario. En el *Sycero,* el usuario se encarga en primer lugar de la definición de archivos y el diseño de la pantalla. Tras ello vienen el «proceso de variables de entrada» (asociando avisos de ayuda y mensajes de error a la entrada del usuario en cada caso), las definiciones de listados y otras varias opciones. Los diversos módulos se unirán entre sí justo antes de la codificación.

Aunque este enfoque es menos amable para el usuario inexperto, ofrece la ventaja de que exige definir cuidadosamente el proyecto antes de codificarlo, ya que de lo contrario no se ejecutará en absoluto. Además, si durante la codificación el generador detecta algún error, detiene el proceso y le informa al usuario acerca del error, de modo que pueda enmendarlo. El proceso de codificación concluye guardando el programa, en archivo binario, que se puede ejecutar desde dentro del entorno del *Sycero.*

Los dos generadores producen un código completamente transportable. De hecho, si el programa *TLO* es suficientemente corto, es posible ejecutar la versión ASCII en una máquina de estándar MSX (en la medida en que se ajuste a las limitaciones de memoria del micro). No obstante, ambos programas incluyen exhaustivas rutinas de detección de errores en los programas que generan, con lo que resultan bastante voluminosos.

Como es común en el software MS-DOS, tanto el *TLO* como el *Sycero* han de estar configurados con el hardware en uso con los programas Install, que se pueden utilizar con máquinas de disco rígido o disco flexible de unidad doble. El *TLO* también debe «certificar» un disco de trabajo, un proceso en cierto modo similar al formateo. A este fin, un disco de trabajo puede ser un disco flexible entero o bien un subdirectorio del disco rígido (este último se certifica como parte del proceso de instalación). La certificación es esencial, porque si durante la corrección del diagrama de flujo el programa no puede detectar una zona de disco de trabajo, será incapaz de hallar el diagrama de flujo en cuestión. Esto significa que el usuario tendrá que volver al principio o bien intentar manipular el código en BASIC. Como siempre, puede protegerse contra esta clase de desastres haciendo frecuentes copias de seguridad del contenido del disco que esté utilizando.

Tanto el *TLO* como el *Sycero* generan una profusa documentación, listando cuestiones tales como las características del diseño de pantalla, los nombres de las variables utilizadas, etc. *Sycero* añade la fecha y la hora en la pantalla, así como en las copias impresas, para que se puedan distinguir, al comprobarlas, las primeras versiones de las posteriores mejoradas.

Tras la instalación, al usuario del *TLO* se le ofrece un Main Dispersal Menu (menú de dispersión principal) de ocho opciones:

Crear un programa	Interrogación
Modificar un programa	Certificar disco nuevo
Modificar un archivo	Resumir codificación
Definir un archivo	Retornar al BASIC

Tras elegir la primera opción y responder en sentido afirmativo a la pregunta: ¿Requerirá archivos su programa?, el usuario ha de definir los archivos ne-

Dos "escritores" de programas
Tanto *The Last One (TLO)* como *Sycero* permiten que el usuario genere programas en BASIC, que se ejecutan con independencia del ordenador anfitrión. Sin embargo, la definición precisa de las necesidades del usuario implica una concienzuda preplanificación antes de que ambos paquetes se puedan utilizar de forma rentable

cesarios con sus campos y tipos de campos (alfabéticos, numéricos o datos). Después de esto, la pantalla visualizará el Flowchart Creation Menu (menú de creación de un diagrama de flujo). En la versión más reciente, hay 20 opciones disponibles, numeradas del 1 al 12 y del 14 al 21 (evidentemente, el autor del *TLO* es supersticioso):

Listar diagrama de flujo	Borrar
Modificar diagrama flujo	Establecer punteros de arch.
Codificar programa	Leer un archivo
Mezclar diagramas flujo	Escribir un archivo
Abortar	Buscar o clasif. un arch.
Entrada por teclado	Mezclar
Visualizar datos	Comprobación de reg.
Bifurcaciones	Borrar archivo
Cálculos	Funciones de base datos
Funciones especiales	Multifunciones

Utilizando estas opciones se puede desarrollar un diagrama de flujo, que podría parecerse a lo siguiente, para un archivo de nombres y direcciones:

Ian McKinnell

1. . Bifurcación a un menú de 3 opciones
2. . Establecer puntero en el final del arch. Direcciones
3. . Entrada por teclado para archivo Direcciones
4. . Escribir datos en archivo Direcciones
5. . Preguntar <¿Has terminado?>. Bifurcar si «No»
6. . Dirigir bifurcación incondicional
7. . Bifurcación a un menú de 3 opciones
8. . Establecer puntero al comienzo del arch. Dirs.
9. . Búsqueda por teclado de archivo Direcciones
10. . Visualizar datos de archivo Direcciones
11. . Preguntar <¿Otra búsqueda?>. Bifurc. si «Sí»
12. . Dirigir bifurcación incondicional
13. . Clasificar archivo Direcciones
14. . Establecer puntero al comienzo del arch. Dirs.
15. . Leer datos de archivo Direcciones
16. . Visualizar datos de archivo Direcciones
17. . Dirigir bifurcación incondicional
18. . Terminar

A B C D E F G H I J K L

Entonces puede empezar la codificación. Hasta este punto, el usuario no ha podido salir del programa sin perder todo el trabajo realizado hasta ese momento. Sin embargo, cuando comienza la codificación se podrá salir y usar la opción Resumir Codificación para volver a comenzar. Durante la codificación, se completan los destinos de las bifurcaciones; el primer menú, por ejemplo, se bifurcará a 2 (escribir datos), 7 (leer datos) o 18 (terminar), y se diseñan las pantallas. Se pueden guardar (SAVE) pantallas, lo que es aconsejable, puesto que se las puede modificar para utilizarlas ulteriormente en cualquier lugar del programa o incluso en otros programas. Sin embargo, ni en el software ni en el manual se le da a esta facilidad la importancia que requiere. En el *Sycero,* las pantallas se guardan (SAVE) automáticamente.

Menú de apertura

El menú de apertura del *Sycero* ofrece 13 opciones que se utilizan más o menos por el orden en que aparecen:

Configuración del sistema
Inicialización
Definición de archivo-campos del sistema
Definición de la pantalla
Proceso de la pantalla
Definición de listado
Proceso de listado
Definición del programa
Generar un programa
Crear un archivo de datos «vivos»
Ejecutar un programa generado
Utilidades
Finalizar la sesión

Cuando se llegue a la opción de generación del programa ya se habrá hecho gran parte del trabajo, y el proceso relativamente simple que sigue exigirá poca intervención, a menos que el software detecte un error obvio. Por supuesto, el sistema no puede deducir sus intenciones, de modo que es imperativo que lo instruya de una manera *precisa.*

La principal diferencia entre los dos generadores es de enfoque. Como se afirma en el propio manual del *Sycero,* «la mejor forma de desarrollar un sistema es utilizar el enfoque de arriba abajo. Comience con un boceto lo más general posible y luego, de forma gradual, vaya rellenando con los detalles desde arriba hacia abajo. Lamentablemente, la mejor forma de entrar los detalles en un generador es la contraria, de abajo arriba».

Al comenzar con un generador de diagrama de flujo, el *TLO* se aproxima más al enfoque clásico de abajo arriba, aunque paradójicamente puede ser que, de este modo, estimule al usuario a comenzar el trabajo con una preplanificación inadecuada. Por el contrario, es virtualmente imposible trabajar con *Sycero* a menos que se haga una preplanificación, si bien hace que resulte más fácil ir revisando las cosas a medida que uno va trabajando.

Tanto *Sycero* como *TLO* ofrecen al usuario una forma conveniente de generar programas en BASIC que se ejecuten independientemente de la máquina anfitriona. A pesar de sus limitaciones, pueden resultar útiles para la construcción de utilidades de bases de datos y programas sencillos de cálculo con preguntas y respuestas.

Tener gancho

La Interface 1 para el Spectrum nos proporciona unas interesantes rutinas que vamos a analizar aquí

La Interface 1 tiene ocho Kbytes de memoria ROM, y ocupa la primera sección de la memoria. Proporciona las rutinas necesarias para gestionar los dispositivos adicionales (como los microdrives); asimismo amplía el intérprete del BASIC para que acepte instrucciones como CAT y FORMAT. Esta ROM suele denominarse ROM Sombra (Shadow ROM).

Hay numerosas versiones diferentes de esta ROM, y existen diferencias de importancia entre la implementación inicial (versión 1) y la actualmente presentada (versión 2). La versión 1 se mostró muy poco eficaz en su empleo de microdrives, y diversas operaciones de las que ofrecía no las podía realizar.

Las versiones siguientes de la ROM son más eficaces en el manejo de los microdrives, además de permitirnos emplear más espacio en un cartucho de microdrive. Asimismo fueron depurados los errores y se incorporaron nuevas facilidades relacionadas con las impresoras en serie.

Estas diferencias de ROMs suelen ser «invisibles» para el usuario, mientras sólo se acceda a la Interface mediante BASIC o con las técnicas consagradas de programación en código máquina. Sin embargo, si usted intenta llamar a la ROM Sombra directamente, con toda probabilidad encontrará dificultades, ya que las rutinas están colocadas en direcciones que difieren según la versión de ROM. Las direcciones específicas de ROM Sombra que damos se refieren a la ROM versión 1.

Pero ¿cómo puede la ROM Sombra ocupar los ocho primeros Kbytes de la memoria si la ROM del BASIC se encuentra justamente en esta zona? La técnica usada es similar al sistema paginado de ROM que emplea el BBC Micro. Siempre que el Z80 vaya a leer una instrucción de las direcciones &08 o bien &1708 con la Interface 1 conectada, la ROM del BASIC es «paginada» y relevada por la ROM Sombra en las operaciones siguientes. Hay una rutina en la ROM Sombra que vuelve a restaurar la ROM del BASIC en su momento.

Antes de que esto suceda, sin embargo, debe inicializarse la Interface 1. Esto se logra después de:

1. Emitir una instrucción NEW una vez acoplada la Interface.
2. Paginar por primera vez la ROM Sombra.

El resultado práctico de este proceso de inicialización es que se ha establecido un nuevo conjunto de variables de sistema. Éstas ocupan la RAM a partir del final de las antiguas variables de sistema y hasta el área de los canales: más de 50 bytes en total. Lo anterior, junto con el establecimiento del mapa del microdrive y demás información necesaria para los distintos dispositivos administrados por la Interface 1, hace que, conectada ésta, el inicio del texto de programas en BASIC se desplace hacia arriba en la memoria. Por esto es muy poco recomendable almacenar el código máquina en las sentencias REM de la línea 1 después de haber conectado la interface (ya que la línea 1 del programa no tiene ahora una posición constante en la memoria).

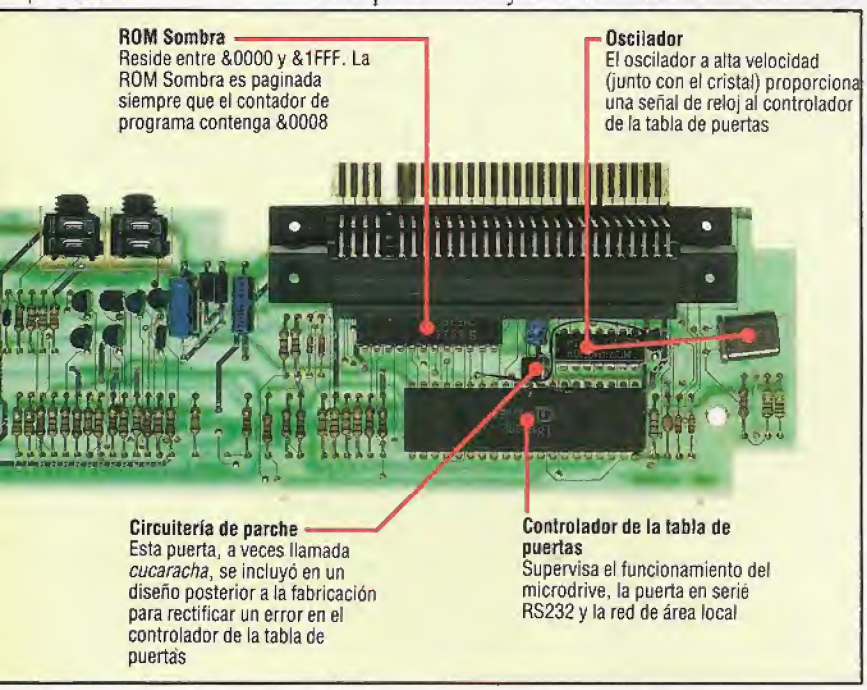
Como ya hemos indicado, la Interface 1 ofrece también instrucciones en BASIC adicionales. Incluso podemos añadir algunas de elaboración propia, como veremos en otro capítulo. Nos limitaremos aquí a examinar someramente la implementación de instrucciones como CAT y FORMAT. En un Spectrum no ampliado, eludirán toda comprobación llevada a cabo por el intérprete del BASIC, por lo que generará normalmente un error. Esto se hace mediante la instrucción RST &08, que accede a la dirección &08. Cuando la Interface 1 es conectada, esta instrucción paginará la ROM Sombra, que proporcionará al enojado fragmento del BASIC unas «segundas gafas» para que pueda ver si se trata de una instrucción que pueda interpretar correctamente. Y si puede, lo hará. De esta manera más o menos funciona una instrucción como la siguiente, que guarda el programa de nombre *Juan* en el microdrive:

```
SAVE *"m";1;"Juan"
```

La parte * de la instrucción genera una condición de error, obligando al control a pasar a la dirección &08, y por tanto a la ROM Sombra, donde tiene lugar la correcta interpretación de la sentencia.

Veamos ahora las rutinas adicionales en lenguaje máquina que incorpora la Interface 1. Examinare-

La procesión por dentro
El dibujo muestra los principales componentes de la placa de circuito impreso correspondiente a la Interface 1. No sólo proporciona facilidades para microdrive, interface serial y LAN, sino que permite la creación de nuevas instrucciones en BASIC definidas por el usuario. En otro capítulo examinaremos estas instrucciones extras definidas por el usuario



mos las que pueden dar en llamarse rutinas para «tareas universales», que son las rutinas no relacionadas con el control del microdrive, la interface serial o las facilidades LAN.

Códigos de enganche

El primer problema es llamar a estas rutinas. La ROM Sombra sólo se activa después de una llamada a alguna de las direcciones anteriormente mencionadas. Esto se resuelve llamando a la dirección &08 mediante las siguientes instrucciones:

```
RST     &08
DEFB    nn
```

donde nn es un valor entre &1B y &32 (los valores fuera de este intervalo generan error). A estos valores nn se les denomina *códigos de enganche (hook codes)*, y cada una de estas llamadas pertenece a una rutina diferente de la ROM Sombra. Los efectos de los códigos de enganche no han cambiado en las distintas versiones de la ROM Sombra. Antes de examinar algunas de estas llamadas, advertiremos varias cosas.

1. Las operaciones del código de enganche suelen afectar a todos los registros, por lo que es necesario proteger los registros que puedan necesitarse después. Además, siempre es bueno guardar el par de registros HL, pues es esencial a la hora de volver al BASIC.
2. Algunas rutinas de los códigos de enganche desactivan las interrupciones. Usted debe reactivarlas si no está seguro.
3. Cuando se usa un código de enganche, ponga el valor &5C3A en el par de registros IY.
4. Antes de emplear un código de enganche, se recomienda asegurarse de que las variables de sistema de la Interface 1 han sido establecidas. Existe un código de enganche que hará esto y vamos a examinarlo antes de nada.

- *Código de enganche 49 (&31):* se encarga de crear, o insertar, las variables de sistema de la Interface 1. Si usted no está seguro de que éstas hayan sido creadas, deberá hacer la siguiente llamada antes de emplear cualquier otro código de enganche:

```
RST     8
DEFB    49
```

Una vez que la ROM Sombra ha ejecutado una rutina de código de enganche, el control pasa a la ROM principal. La ROM Sombra también nos proporciona algunas rutinas que están presentes en la ROM principal del BASIC de una manera menos adecuada. Éstas tratan de entradas/salidas a pantalla desde el teclado.

- *Código de enganche 27 (&1B):* provoca una espera hasta que se digita en el teclado un carácter, y pone en el registro A el código del carácter correspondiente a la tecla pulsada (obsérvese que esto no aparece en pantalla).

- *Código de enganche 32 (&20):* inspecciona el teclado en el momento de ser llamado e indica (por medio del estado del flag C) si se ha pulsado una tecla o no. Esta rutina no espera hasta que se pulse una tecla. Si ha sido pulsada, el flag C se pone a 1, y en caso contrario a cero. Es fundamental que se activen las interrupciones antes de llamar a los códigos de enganche 27 y 32, dado que la inspección del teclado en el Spectrum se gestiona mediante interrupciones.

- *Código de enganche 28 (&1C):* se encarga de la impresión de un carácter (que conoce gracias al código ASCII previamente contenido en el registro A) en la corriente 2, que suele ser la parte superior de la pantalla de televisión. El empleo de esta rutina, junto con la del código 27, se ilustra en el siguiente fragmento en lenguaje máquina. Si usted utiliza este programa, observe que es un bucle indefinido, y que tendrá que desconectar el ordenador para salir de él.

```
        ;llama rutina ROM Sombra - ponga atención!
210000  ld    hl,ADDRESS        ;inserta direccion de rutina
22ED5C  ld    (23789),hl        ;pone direccion en variable de sis-
CF      rst   #08               tema
32      defb  #32               ;opera
```

- *Código de enganche 31 (&1F):* es semejante al código 28, sólo que escribe el carácter (cuyo código ASCII está en el registro A) en la corriente 3, que suele ser la impresora ZX.

- *Código de enganche 50 (&32):* se trata del código de enganche final para tareas universales, y lo vamos a examinar con más detención. En las especificaciones del fabricante se le rotula, no sin un cierto misterio, Sinclair Research Use Only (sólo para uso de Sinclair Research). Gracias a él podemos llamar a una rutina de la ROM Sombra a una dirección determinada desde la ROM principal, sin tener que usar códigos de enganche. A causa de las diferencias entre las distintas versiones de la ROM Sombra todo programa que emplee esta llamada para acceder a las rutinas de aquélla se vendrá abajo si se usa en una versión diferente. Los detalles que damos aquí son, pues, a efectos de completitud.

La dirección de la rutina de la ROM Sombra a la que deseamos acceder se encuentra en dos bytes dentro de las variables de sistema mostradas por la Interface 1. La dirección de la variable en cuestión está en las posiciones 23789 y 23790. Seguidamente se realiza el código de enganche, lo que se conseguirá con el siguiente fragmento en código máquina:

```
        ;espera el carácter, imprime en corriente 2
        ;cod. de eng. &1B y &1C
CF      start:  rst   #08       ;establece variables sistema
31              defb  #31       ;... Interface 1
FB      loop:   ei              ;ei, por si acaso
CF              rst   #08       ;espera un caracter
1B              defb  #1B       ;... del teclado
CF              rst   #08       ;ahora el caracter esta en A...
1C              defb  #1C       ;... y lo imprime
18F9            jr    loop      ;vuelta hasta el infinito
```

Desde luego que no es una llamada extremadamente útil a menos que se conozcan las funciones de las distintas rutinas de la ROM Sombra y con qué versión de dicha ROM se está trabajando. No obstante, deberemos examinar esta llamada con mayor detención cuando tratemos cómo se emplea la Interface 1 para añadir nuevas instrucciones al BASIC. En el próximo capítulo, de momento, examinaremos aquellos códigos de enganche que son específicos de las operaciones del microdrive.

Interacción interestelar

Aventura clásica
The hitchhiker's guide to the galaxy es la última de una larga serie de aventuras de Infocom, de muchísimo éxito, que se inició con la *Trilogía de Zork*, tres juegos llenos de tesoros, magia y misterio que se desarrollaban en el «imperio Zork»

Jill Furmanovsky

Una fértil imaginación
Douglas Adams, autor de *The hitchhiker's guide to the galaxy* (Guía de la galaxia para el autostopista). Escrita originalmente para la radio, desde entonces la serie ha sido publicado por Pan Books, registrando en el mundo unas ventas extraordinarias

¿Qué diferencias esenciales existen entre un juego basado en disco y otro basado en cassette? Veámosla a través del análisis de un juego de aventuras recién aparecido

The hitchhiker's guide to the galaxy: Para la gama de ordenadores Apple, Apricot, IBM y Atari. Pronto habrá una versión para el Commodore 64
Distribuido por: Softsel, Softsel House, Sion Gate Way, Great West Road, Brentford, Middlesex, TW18 9DD, Gran Bretaña
Palanca de mando: No se necesita
Formato: Disco

Los juegos de aventuras requieren enormes cantidades de almacenamiento de datos y son ideales para los sistemas basados en disco, en los cuales las descripciones de nuevos escenarios, mensajes y rutinas de instrucciones se pueden cargar desde disco cuando es necesario. Esta situación siempre ha representado un problema para los programadores de aventuras europeos, quienes han debido escribir juegos para un mercado que tradicionalmente ha rechazado el almacenamiento en disco en favor del realizado en cassette, de menor capacidad. Por este motivo, casi todo el software de aventuras europeo está basado en RAM y, por consiguiente, bastante limitado, aunque los programadores se han vuelto expertos en explotar el limitado espacio disponible. En particular, se ha frenado el desarrollo de «personajes interactivos» (que requieren grandes cantidades de datos), así como la introducción de juegos con abundante vocabulario.

En Estados Unidos, sin embargo, la situación es muy diferente. Todos los micros populares personales tienen fácil acceso a sistemas de disco, y la disponibilidad de ingresos, que registra niveles por lo general más elevados que los de los usuarios europeos, ha significado que tales sistemas por lo general los adquieran los usuarios noveles. El mercado, por lo tanto, ha sido ideal para el desarrollo de complejo software de aventuras o, como algunos prefieren llamarlo, de «ficción interactiva». La casa de software norteamericana Infocom ha sido el líder en juegos para este mercado.

El último producto de Infocom es representativo de los elevados estándares que actualmente se esperan de la compañía. Se trata de *The hitchhiker's guide to the galaxy* (Guía de la galaxia para el autostopista), escrito por Douglas Adams en colaboración con los programadores de Infocom para el Apple IIe y la gama de ordenadores Atari (aparecerá también una versión para el Commodore 64). Exige una única unidad de disco y muchísima paciencia.

El juego se basa, con bastante fidelidad, en la serie escrita originalmente para la radio por Douglas Adams, e incorpora personajes tales como Ford Prefect, Zaphod Beeblebrox, numerosos alienígenas y, por supuesto, Arthur Dent, el antihéroe de la historia, quien un buen día se encuentra arrancado de su existencia suburbana y llevado secretamente a la inmensidad del espacio en la cabina de carga de un Vogon Starcruiser.

El primer punto fuerte del juego, y el más obvio, es el analizador gramatical, la parte del programa que acepta e interpreta la entrada del usuario. Los analizadores gramaticales de Infocom pueden distinguir adjetivos, adverbios y preposiciones, además de los más tradicionales verbos y sustantivos a los que se limitan la mayoría de los juegos de aventuras europeos. Además, el vocabulario es sumamente amplio (entre mil y dos mil palabras) y la entrada puede tener numerosos formatos diferentes. Por ejemplo, usted puede entrar una instrucción directa (como Beber la cerveza), una instrucción múltiple (Coger el paquete y después guardárselo en el bolsillo), o una pregunta directa (¿Dónde estoy?).

El jugador se puede dirigir a los personajes simplemente diciendo sus nombres, como en Ford, ¿dónde estamos? o Marvin, vete. Incluso aunque el ordenador no comprenda exactamente lo que se espera de él, el programa por lo general aparecerá con una respuesta aceptable: una enorme mejora respecto a respuestas tradicionales como No puedes hacer eso o, simplemente, No comprendo.

Otras técnicas de programación incorporadas en este juego incluyen la provisión de «contenedores» (objetos que pueden retener otros objetos), una facilidad que con frecuencia falta en los juegos europeos. Otras provisiones son «objetos globales» (tales como «suelo», «pared», etc.), que pueden estar presentes en muchos escenarios diferentes, y «vehículos», contenedores que pueden retener y transportar al jugador de un escenario a otro. Es interesante observar, no obstante, que la mayoría de estas facilidades, junto con el enorme vocabulario y el eficaz analizador gramatical, son resultado del mayor almacenamiento de datos permitido por los discos y no de una programación más inteligente.

El desensamblado de un juego producido por Infocom revela un diseño muy complejo del argumento y el guión, pero poco en cuanto a lo que hemos dado en llamar inteligencia artificial. Entre algunos programadores de aventuras europeos existe la creencia generalizada de que la exuberancia de posibilidades que ofrece la programación basada en disco ha vuelto algo perezosos a sus colegas norteamericanos y que, cuando el mercado europeo abandone las cassettes, el hábito de aplicar las técnicas de compresión desarrolladas para los juegos basados en RAM traerá como lógica consecuencia la producción de programas de calidad superior.

Tony Sleep

¿Cuál es la expresión de tu rostro?
El sistema WISARD, de Igor Aleksander y desarrollado en el Imperial College de Londres, utiliza una cuadrícula de 512 por 512 y un megabyte de RAM para poder reconocer patrones visualizados en una pantalla de televisión. Es tan sensible que se le puede enseñar a distinguir entre rostros sonrientes y rostros serios

Reconocer un gesto

En esta ocasión analizaremos un método que explora patrones en busca de rasgos significativos

Los llamados sistemas de reconocimiento de patrones *de abajo arriba* intentan extraer rasgos significativos de una imagen dada y reducirla a un patrón más simple y de un mayor nivel de abstracción. Un método para hallar tales rasgos en un patrón consiste en utilizar operadores locales para explorarlo. Cada operador local explora una pequeña superficie cada vez y multiplica las intensidades de gris de los puntos bajo el operador por factores de estimación. Estos factores están diseñados para producir marcadores elevados cuando se localiza el rasgo buscado. Mediante el empleo de numerosos operadores locales diferentes, se pueden extraer las posiciones de rasgos importantes, tales como líneas horizontales, verticales y oblicuas o parches de luz y oscuridad.

El programa WISARD, de Igor Aleksander, que describimos en el capítulo anterior, utiliza el principio de *generación de dirección de memoria* desde un patrón con el objeto de recordar el patrón y reconocerlo posteriormente.

La gran ventaja del WISARD reside en que todos los bancos de RAM se direccionan en paralelo. Por tanto, puede trabajar a gran velocidad con imágenes de televisión muy nítidas. El programa en BASIC que ofrecemos como ejemplo simula secuencialmente un sistema de este tipo. Es mucho mas lento, pero demuestra claramente los principios involucrados.

Para su labor de diferenciación el sistema utiliza séxtuplos (son menos potentes que los óctuplos pero ahorran espacio) y requiere 20 480 bits de RAM, o 2 560 bytes. Ello se debe a que hay 40 séxtuplos, cada uno de los cuales puede estar en 64 estados, y, por tanto, direccionar un banco de RAM de 64 posiciones.

Cada posición contiene ocho bits, lo que permite al sistema diferenciar ocho distintas clases de entrada.

El programa posee dos fases: la primera es una etapa de entrenamiento en la cual se proporcionan al ordenador ejemplos de cada diferente clase de entrada. La segunda fase implica el reconocimiento de un patrón a través de la comparación con ejemplos aprendidos previamente. Por ejemplo, supongamos que durante la fase de entrenamiento el séxtuplo número 20 da como respuesta 110010 (50) cuando está presente el patrón de clase 4. En este caso, el cuarto bit de la posición 50 del banco de RAM 20 se establecerá en 1. Si durante la fase de reconocimiento el diferenciador vuelve a responder con otro 50, encontrará al cuarto bit establecido en

la dirección especificada, lo que tiende a confirmar la presencia del patrón número 4.

Para retener el patrón de datos se utiliza una serie de bytes que comienza en D, y toda la matriz se borra al principio de cada ejecución. Esto se consigue mediante la subrutina 1000. La subrutina 2000 permite al usuario «pintar» en la pantalla imágenes sencillas de 16 por 16, utilizando las teclas U, D, L y R para mover, y usando el cáracter * y la barra espaciadora para crear el patrón.

El programa se puede modificar de modo que el usuario pueda cargar y guardar la matriz D en disco o cinta. Tal como está, la primera vez que ejecute el programa, habrá de enseñarle una tarea de clasificación a partir de cero, de modo que, en este sentido, está en libertad de introducir cualquier mejora que considere necesaria. Por supuesto, lo ideal sería que el sistema estuviera unido al mundo exterior a través de un convertidor de analógico a digital fijado en algún tipo de cámara.

Observe que la asignación de pixels a los séxtuplos se realiza al azar, pero que se ha de poder repetir. La imagen del patrón inicialmente se retiene en una matriz bidimensional, I(,), utilizando un uno para representar un asterisco y un cero para representar un espacio. Durante la fase de entrenamiento se generan direcciones desde elementos de I(,) seleccionados al azar. El bit apropiado (según la clase que se esté aprendiendo) se establece en la línea 210. Durante la fase de reconocimiento se debe generar la misma secuencia de direcciones seleccionadas al azar, para poder comparar el patrón que se está comprobando con las clasificaciones aprendidas previamente.

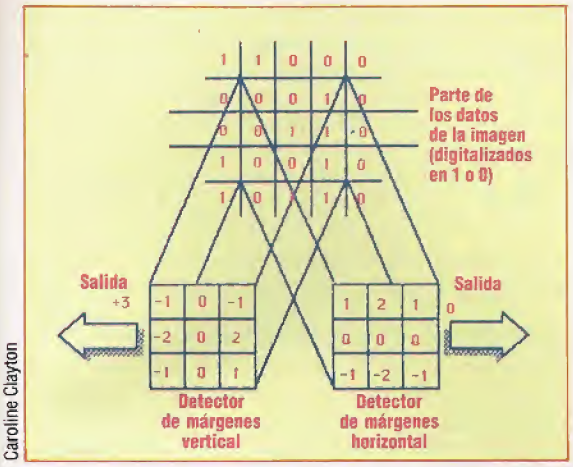
Programa de reconocimiento de patrones

BBC Micro:

```
10 REM ********************************************************
30 REM ** RECONOCEDOR DE PATRONES **
40 REM ** TIPO WISARD PARA EL BBC **
50 REM ********************************************************
55 MODE 7
56 SX%=40: DS%=6: RB%=2^DS%
57 MEMO%=SX%*RB%
60 DIM I%(16,16), CS%(7)
64 DIM D% MEMO%
65 @%=4: REM formato o/p
66 REM D% es una matriz de bytes (BBC)
70 REM - - I% es matriz de imagen, D% es matriz direcciones
80 REM - - Nota: I% contiene 0 o 1
85 PRINT "Tamaño total de RAM = ";MEMO%," bytes."
88 GOSUB 1000: REM limpiar matriz D%
90 REM - - Primero la fase de entrenamiento:
92 MOOD$="Entrenamiento"
95 INPUT "De que clase es (0..7) ",C%
96 IF C% <0 OR C%>7 THEN GOTO 90
100 FOR I%=1 TO 16:FOR J%=1 TO 16: I%(I%,J%)=0
101 NEXT:NEXT
110 REM - - Ahora tomar la imagen pintada por el usuario:
120 GOSUB 2000
130 R=RND(-1):REM valor al azar
140 FOR I%=1 TO SX%
150 A%=RB%*(I%-1)
155 REM A% es la direccion de base del banco de RAM.
160 FOR J%=0 TO (DS%-1)
170 R1%=INT(RND(1)*16)+1
180 R2%=INT(RND(1)*16)+1
190 A%=A%+I%(R1%,R2%)*2^J%
200 NEXT J%
202 REM R1% y R2% son coordenadas «aleatorias».
210 PROCdset(A%,C%)
220 NEXT I%
230 INPUT "Otra sesion de entrenamiento (S/N)?",A$:IF A$="S" THEN GO TO
    90
233 IF A$="s" THEN GOTO 90
235 MOOD$="Reconocimiento"
236 REM - - Ahora la Fase de Reconocimiento:
240 FOR I%=1 TO 16: FOR J%=1 TO 16
244 I%(I%,J%)=0
245 NEXT :NEXT
250 REM - - Obtener una imagen para clasificacion:
260 GOSUB 2000
270 R=RND(-1)
280 FOR C%=0 TO 7:CS%(C%)=0: NEXT
290 FOR I%=1 TO SX%
300 A%=RB%*(I%-1)
310 FOR J%=0 TO DS%-1
320 R1%=INT(RND(1)*16)+1
330 R2%=INT(RND(1)*16)+1
340 A%=A%+I%(R1%,R2%)*2^J%
350 NEXT
355 FOR C%=0 TO 7
360 IF DNdget(A%,C%) > 0 THEN CS%(C%)=CS%(C%)+1
370 NEXT C%
380 NEXT I%
382 CX%=0
385 FOR C%=0 TO 7
388 PRINT "La clase";C%;" posee un marcador de
    ";CS%(C%)/SX%*100;SPC(8)
390 IF CS%(C%)>CS%(CX%) THEN CX%=C%
400 NEXT C%
404 PRINT "Lo mas probable es que la clase sea la numero ";CX%
410 INPUT "Quieres clasificar otra imagen (S=Si) ";A$
420 IF A$="S" OR A$="s" THEN GOTO 240
440 PRINT: PRINT "Adios!"
444 END
999 :
1000 REM - - Inicializacion de memoria:
1010 FOR I%=0 TO MEMO%
1020 REM - - usa subindices de byte (?):
1030 ?(D%+I%)=0
1040 NEXT
1050 RETURN
1199 :
1200 DEF PROCdset(A%,C%)
1210 REM - - - - - - - -
1220 ?(D%+A%) = (D%?A%) OR 2^C%
1230 ENDPROC
1240 REM establece un bit de la matriz D%
1244 :
1250 DEF FNdget(A%,C%)
1260 REM - - - - - -
1270 =(D%?A%) AND 2^C%
1280 REM extrae bit de la matriz D%
1288 REM A% es direccion: C% es 0..7
1290 :
2000 REM - - Rutina formacion-imagen:
2010 CLS:PRINT TAB(0,20);"(Usar: U, D, L, R, X * o espacio"
2012 PRINT "para definir la imagen.)"
2013 PRINT MOOD$; "fase."
2015 PRINT TAB(1,1);
2020 T%=0;A$=" "
2030 H%=1: V%=1
2040 REPEAT C$=INKEY$(222)
2050 IF C$="U" THEN V%=V%-1
2060 IF C$="D" THEN V%=V%+1
2070 IF C$="L" THEN H%=H%-1
2080 IF C$="R" THEN H%=H%+1
2090 IF H% > 16 THEN H%=1
2095 IF H% < 1 THEN H%=16
2100 IF V% > 16 THEN V%=1
2105 IF V% < THEN V%=16
2110 IF C$=" " THEN T%=0:A$=C$
2120 IF C$="*" THEN T%=1:A$=C$
2130 I%(H%,V%)=T%
2140 PRINT TAB(H%,V%);A$;
2150 UNTIL C$="X"
2155 GOSUB 2200: REM Volver a visualizar
2160 RETURN
2170 :
2200 REM - - Rutina de visualizacion:
2210 FOR H%=1 TO 16
2220 FOR V%=1 TO 16
2230 IF I%(H%,V%)>0 THEN PRINT TAB(H%,V%);"*" ELSE PRINT
    TAB(H%,V%);" ";
2240 NEXT: NEXT
2250 PRINT TAB(0,17);
2260 RETURN
2270 :
2300 REM las instrucciones para "pintar" son:
2301 REM U & D para arriba y abajo;
2302 REM L & R para izquierda y derecha;
2303 REM * & espacio para On y Off;
2304 REM X para salir (imagen hecha).
```

Commodore 64:

```
10 REM **** RECONOCEDOR ****
20 REM **** DE PATRONES CBM 64 ****
30 FOR I=1 TO 25: DW$=DW$+CHR$(17):NEXT I
56 SX=40:DS=6:RB=2^DS
57 ME=SX*RB
60 DIM I(16,16),C(8)
64 D=12*4096:REM USAR $C000 PARA MATRIZ D
85 PRINT "TAMAÑO TOTAL DE RAM=";ME;" BYTES"
88 GOSUB 1000:REM LIMPIAR MATRIZ D
90 REM **** FASE DE ENTRENAMIENTO ****
92 M$="ENTRENAMIENTO"
95 INPUT"DE QUE CLASE ES (1...8)";C
96 IF C<1 OR C>8 THEN 95
100 FOR I=1 TO 16:FOR J=1 TO 16:I(I,J)=0
101 NEXT J:NEXT I
```



Caroline Clayton

Ojo óctuplo
Conformando la base del sistema WISARD para reconocimiento de patrones hay grupos de ocho píxels conectados arbitrariamente a ocho registros de bits. Cada óctuplo corresponde a un banco de 256 posiciones. Durante la fase de entrenamiento, el valor del óctuplo forma una dirección dentro de su banco de RAM, según el patrón de píxels, y en esta dirección se graba un 1. Si durante la fase de reconocimiento está presente el mismo patrón, se generará la misma dirección del banco de RAM y el 1 presente significará el reconocimiento

Operadores locales
En la ilustración de la izquierda vemos un ejemplo del tipo de proceso de bajo nivel que puede emplear un sistema de reconocimiento de patrones de abajo arriba. Las cuadrículas de tres por tres se denominan *operadores locales* y exploran los datos de la imagen y registran marcadores elevados en las transiciones de bordes, dando, por lo tanto, un boceto primario de la imagen basado en líneas rectas. Los operadores se mueven a través de los datos de la imagen y cada número se multiplica por la intensidad de niveles de gris de la sección del patrón sobre la cual esté colocado. La suma de los productos da marcadores altos cuando el rasgo deseado está presente, y marcadores bajos en caso contrario. Cada uno de los nueve números de un operador local es un factor de estimación; el patrón de estas estimaciones determina los rasgos que se detectarán

```
110 REM **** AHORA OBTENER IMAGEN PINTADA POR USUARIO ****
120 GOSUB 2000: REM CONSTRUCTOR DE IMAGENES
130 R=RND(-1):REM VALOR ALEATORIO
140 FOR I=1 TO SX
150 A=RB*(I-1):REM CALC DIRECCION DE BANCO RAM
160 FOR J=0 TO (DS-1)
170 R1=INT(RND(1)*16+1):R2=INT(RND(1)*16+1)
190 A=A+I(R1,R2)*2^J
200 NEXT J
210 POKE(D+A),PEEK(D+A) OR 2^(C-1))
220 NEXT I
230 INPUT"OTRA SESION DE ENTRENAMIENTO (S/N)?";A$
233 IF A$="S" THEN 90: REM REPETIR
234 REM **** FASE DE RECONOCIMIENTO ****
235 M$="RECONOCIMIENTO"
240 FOR I=1 TO 16:FOR J=1 TO 16
244 I(I,J)=0
245 NEXT J:NEXT I
250 GOSUB 2000: REM TOMAR UNA IMAGEN PARA CLASIFICACION
270 R=RND(-1)
280 FOR C=1 TO 8:C(C)=0:NEXT C
290 FOR I=1 TO SX
300 A=RB* (I-1)
310 FOR J=0 TO DS-1
320 R1=INT(RND(1)*16+1):R2=INT(RND(1)*16+1)
340 A=A+I(R1,R2)*2^J
350 NEXT J
355 FOR C=1 TO 8
360 IF (PEEK(D+A)ANS 2^(C1))>0 THEN C(C)=C(C)+1
370 NEXT C
380 NEXT I
381 PRINT CHR$(147)
382 CX=0:X=0:Y=17:GOSUB 3000
385 FOR C=1 TO 8
388 PRINT"LA CLASE "C" TIENE UN MARCADOR DE ";C(C)/SX*100
390 IF C(C) > C(CX) THEN CX=C
400 NEXT C
404 PRINT"LO MAS PROBABLE ES QUE SEA DE LA CLASE NUMERO";CX
410 INPUT "CLASIFICAR OTRA IMAGEN (S/N)";A$
420 IF A$="S" THEN 240: REM REPETIR
444 END
999 :
1000 REM ***** INIC MEMORIA *****
1010 FOR I=0 TO ME:POKE(D+1),0:NEXT I
1030 RETURN
1040 :
2000 REM **** CONSTRUCTOR DE IMAGENES ****
2005 PRINT CHR$(147)
2010 X=0:Y=20:GOSUB 3000:PRINT"USAR U,D,L,R,X,* 0 ESPACIO"
2012 PRINT"PARA DEFINIR LA IMAGEN"
2013 PRINT "FASE";M$
2015 X=1:Y=1:GOSUB 3000
2020 T=0: A$="  ":H=1:V=1
2040 GET C$
2050 IF C$="U" THEN V=V-1:IF V<1 THEN V=16
2060 IF C$="D" THEN V=V+1:IF V>16 THEN V=1
2070 IF C$="L" THEN H=H-1:IF H<1 THEN H=16
2080 IF C$="R" THEN H=H+1:IF H>16 THEN H=1
2110 IF C$="  " THEN T=0:A$=C$
2120 IF C$="*" THEN T=1:A$=C$
2130 I(H,V)=T
2140 X=H:Y=V:GOSUB 3000:PRINT A$;
2150 IF C$<>"X" THEN 2040
2155 GOSUB 2200
2160 RETURN
2170 :
2200 REM **** RUTINA VISUALIZACION ****
2210 FOR H=1 TO 16
2220 FOR V=1 TO 16
2230 X=H:Y=V:GOSUB 3000:IF I(H,V)=0 THEN PRINT":":GOTO 2240
2235 PRINT"  ";
2240 NEXT V:NEXT H
2250 X=0:Y=17:GOSUB 3000
2260 RETURN
2270 :
3000 REM **** INSTRUCCION TAB ****
3010 PRINT CHR$(19);TAB(X);LEFT$(DW$,Y);
3020 RETURN
```

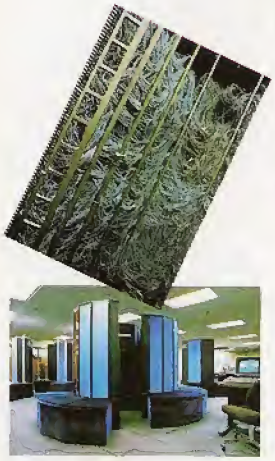
Complementos al BASIC

Spectrum:

El Spectrum no soporta las instrucciones sobre bits AND y OR. En consecuencia, hemos de escribir cortas rutinas en código máquina para llevar a cabo estas operaciones. Éstas se ofrecen en forma de un cargador de BASIC. Introduzca estos cambios en la versión para el Commodore 64:

```
30 CLEAR 49999:GOSUB 4000
64 LET D=50018
130 RANDOMIZE 1
210 LET X1=PEEK(D+A):LET Y1=2↑(C-1):
    GOSUB 6000
215 POKE(D+A),R1
270 RANDOMIZE 1
360 LET X1=PEEK(D+A):LET Y1=2↑(C-1):
    GOSUB 5000
365 IF R1>0 THEN C(C)=C(C)+1
381 CLS
2005 CLS
3010 PRINT AT Y,X;
4000 FOR I=50000 TO 50017
4010 READ A:POKE I,A:NEXT I
4020 RETURN
4030 DATA 62,0,14,0,161,6,0,79,201
4040 DATA 62,0,14,0,177,6,0,79,201
5000 REM **** AND ****
5010 POKE 50001,X1:POKE 50003,Y1
5020 LET R1=USR 50000:RETURN
6000 REM **** OR ****
6010 POKE 50010,X1:POKE 50012,Y1
6020 LET R1=USR 50009:RETURN
```

Ventajosa comparación



La forma del favorito
El Cray-1 (arriba) ofrece al usuario un enorme poder de proceso bajo el control de un ordenador frontal como un IBM, un DEC u otro similar; la arquitectura del Cray-1 (abajo) configura 32 Mbytes de memoria principal y 4 888 bytes de espacio para registros, contenidos en un sistema de 3 400 placas de circuito impreso separadas y más de 90 km de cable para conectarlas

Centraremos nuestra atención en el Cray-1, superordenador capaz de ejecutar aplicaciones de proceso masivo de información

La «potencia» de un ordenador, en términos llanos, es el producto de su tamaño de palabra, su velocidad de transferencia de datos, el tamaño de su memoria principal y la velocidad de ciclos de máquina de su unidad central de proceso. En los microordenadores, las funciones principales de la CPU tradicional están empaquetadas en un único microprocesador. Entre estos microprocesadores se incluyen el difundido Z80, el Intel 8088 y el 8086 y el Motorola M68000. Todos utilizan tecnología MOS (*metal oxide semiconductor*) para los circuitos lógicos y la memoria *on-chip*. Los datos se transfieren y se procesan en paralelo ya sea de ocho en ocho o bien de 16 en 16. En los microprocesadores las frecuencias de reloj van de 1 MHz a 12 MHz.

Las especificaciones como éstas, aun siendo notables, hacen que los ordenadores basados en microprocesador sean incapaces de afrontar la colosal cantidad de proceso de datos que requieren aplicaciones tales como el proceso de imágenes, la dinámica de fluidos y el pronóstico meteorológico.

Supongamos que usted está haciendo una película con gráficos animados generados por ordenador, y que la imagen requiere una resolución de 6 000 por 6 000 pixels y 24 fotogramas por segundo. Puesto que los gráficos son animados, la posición de cada punto de la imagen puede moverse entre los fotogramas, de modo que se habrá de calcular la posición de cada punto para cada fotograma. Eso se traduce en 864 millones de cálculos por segundo, y es probable que cada cálculo sea bastante complejo, implicando docenas o centenares de instrucciones en código máquina. Esto añade hasta billones de instrucciones por segundo. El recuadro *Pruebas de tiempo* muestra una comparación aproximada del tiempo que consume producir una secuencia de película de 10 minutos en un superordenador y en un micro basado en el Z80.

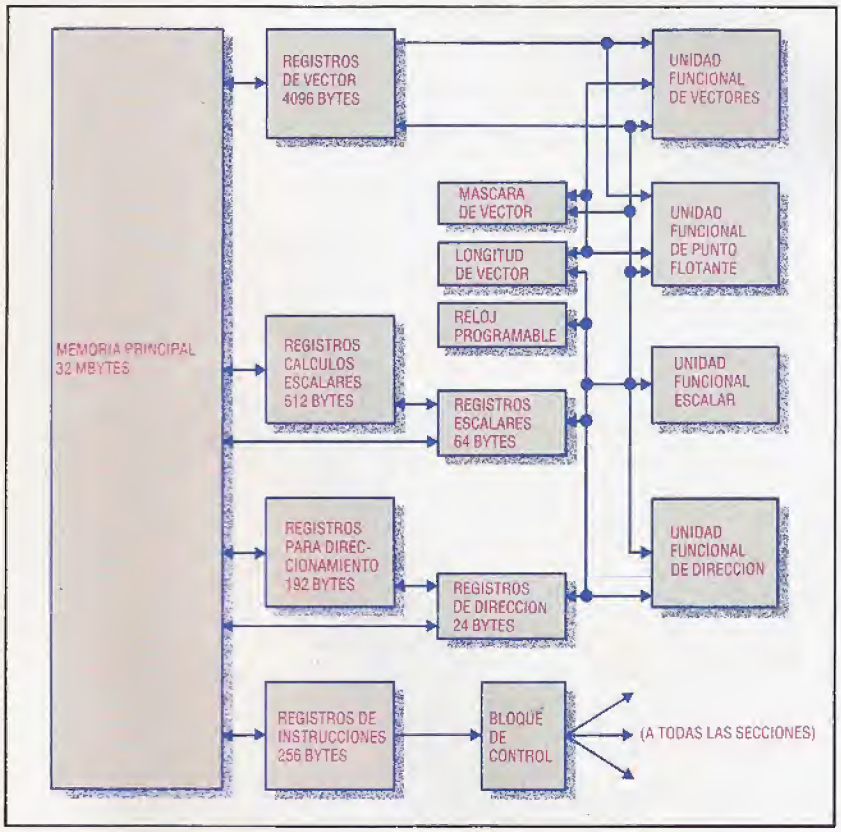
Los superordenadores, tal como tiende a llamarse a los mayores ordenadores centrales, trabajan de forma muy parecida a los microordenadores; las instrucciones y los datos se traen de la memoria, el procesador manipula los datos según sus instrucciones y los resultados se almacenan en la memoria. La principal diferencia radica en la escala y la velocidad a la cual se llevan a cabo estas operaciones, y en la forma en que se construyen los componentes de los ordenadores.

Ilustraremos un superordenador «típico» refiriéndonos al Cray-1 S/4400, fabricado por Cray Research. El primer Cray-1 se instaló en 1976, sólo cuatro años después de la creación de la compañía. Desde entonces se ha establecido como el superordenador más famoso (y uno de los más populares).

La CPU del Cray-1 ocupa una cabina semicircular de 6,5 pies (1,98 m) de altura, de forma similar al de un sofá curvo (debajo de los «asientos» está el sistema de refrigeración y las fuentes de alimentación). Obtiene su velocidad mediante el empleo de lógica de semiconductores bipolares (los semiconductores bipolares son transistores «normales», al contrario que los MOS, CMOS, NMOS, FET —*field effect transistor*— y otros tipos de semiconductores). La lógica bipolar y la memoria vienen en más de 200 000 circuitos integrados en 3 400 placas de circuito impreso separadas, para conectar las cuales se utilizan más de 90 kilómetros de cable.

El tamaño de palabra que se utiliza para los cálculos es de 64 bits de ancho (la cantidad de datos que se procesan a la vez es ocho veces más que en un Z80 de ocho bits) y la velocidad del reloj del sistema es de 80 MHz (80 millones de ciclos por segundo). Una suma típica de 64 bits lleva sólo 37,5 nanosegundos. Una suma de ocho bits en un Z80 operando a 4 MHz (p. ej., ADD A,n) lleva 1,75 microsegundos.

La memoria principal del Cray es una parte inte-



gral de la CPU. Contiene 32 megabytes organizados en 4 194 304 palabras. La memoria puede transferir hasta 2 560 millones de bytes por segundo.

Como es de suponer, el conjunto de registros de la CPU del Cray es impresionante. Hay 72 registros de dirección (cada uno de 24 bits de ancho), 72 registros escalares (cada uno de 64 bits de ancho) y ocho registros de vector (cada uno de 64 palabras de ancho). El espacio total de registros de la CPU del Cray es, por lo tanto, de 4 888 bytes (el del microprocesador Z80 es de apenas 26).

A diferencia de la mayoría de los otros ordenadores, que son unidades independientes, los ordenadores Cray están diseñados para usarse como extensiones de miniordenadores u ordenadores centrales ya existentes. El ordenador frontal (*front end*) actúa como una interface, tomando la entrada de los terminales o lectores de tarjetas, y alimentando la salida a periféricos tales como impresoras o unidades de almacenamiento en cinta magnética.

Entre la CPU del Cray y el ordenador central frontal se halla el subsistema de E/S Cray, diseñado para facilitar las inmensas demandas de uso de la CPU, y una interface frontal, diseñada para adecuar el sistema Cray a las características específicas del IBM, DEC, Data General u otros ordenadores frontales. El subsistema de E/S Cray consta de entre dos y cuatro procesadores de E/S, cada uno de los cuales es un potente miniordenador en toda la regla.

Esta descripción de la arquitectura del Cray-1 apenas si toca la superficie, aunque el diagrama de bloques de la CPU tal vez lo ayude a hacerse una idea de su nivel de sofisticación. No hay espacio suficiente para hablar del juego de instrucciones, las 13 unidades funcionales, como se las llama, que pueden operar en paralelo, ni del proceso de vectores que permite operar, mediante una única instrucción, sobre hasta 64 pares de operandos. Basta con decir que el Cray es una máquina *muy* potente.

Aplicaciones específicas

Pronóstico meteorológico
Los pronósticos meteorológicos son ahora el resultado de la recogida de datos meteorológicos internacionales, comunicación por satélite y modelación por ordenador. Un «modelo» por ordenador del tiempo en el mundo implica cientos de millones de cálculos en vastas matrices de datos, necesitándose las predicciones en cuestión de horas. Sólo ordenadores inmensamente potentes como el Cray pueden manipular esta cantidad de datos con suficiente rapidez

Dinámica de fluidos
La dinámica de fluidos juega un importante papel en el diseño de coches eficaces desde el punto de vista del combustible, los sistemas refrigerantes de las centrales de energía nuclear, la ingeniería aeronáutica y muchas otras aplicaciones. El análisis y la predicción del movimiento de los fluidos exige analizar montones de datos. Los problemas se agravan si se necesitan resultados en tiempo real. La cuestión estriba en que cada partícula de un fluido (y hay incontables trillones de ellas) tendrá un efecto sobre todas las otras partículas del sistema cuando éste se mueva. Por consiguiente, calcular el comportamiento de un sistema completo de fluidos exige un poder informático masivo

Previsión económica
Construir modelos de economía es notablemente difícil. Al igual que en la dinámica de fluidos, un pequeño cambio en cualquier parte del sistema puede tener efectos que se extiendan a través del resto del sistema. Intentar crear un modelo económico de todo el mundo es apenas viable, pero aun los modelos simplificados poseen una complejidad abrumadora. Se necesitan ordenadores rápidos y muy veloces para que los analistas no hayan de esperar durante meses los resultados

Pruebas de tiempo

Para comparar el poder de proceso de un superordenador Cray-1 y un ordenador personal basado en el Z80, como el Amstrad CPC-464, consideremos los problemas que entraña hacer una secuencia de película de 10 minutos de gráficos animados y en alta resolución. Supongamos que se requiere una resolución de imagen de 6 000 por 6 000 pixels y 24 fotogramas por segundo. Supongamos, asimismo, que es necesario calcular individualmente cada pixel de cada nuevo fotograma, y que para la versión Z80 se precisarán 100 instrucciones en código máquina, con un tiempo de ejecución medio de 19 ciclos de reloj (4,75 microsegundos). Para el Cray-1, con sus poderosas instrucciones de proceso de vectores, diremos que se necesitan 25 instrucciones en código máquina (para pecar de conservadores) con un tiempo de ejecución promedio de cuatro ciclos de reloj (50 nanosegundos). Un Z80 tardaría $6000^2 \times 24 \times 60 \times 10 \times 100 \times 4{,}75 \times 10^{-6} = 2{,}4624 \times 10^8$ segundos, o 7,8 años. Un Cray-1 tardaría $6000^2 \times 24 \times 60 \times 10 \times 25 \times 50 \times 10^{-9} = 6{,}48 \times 10^5$ segundos, o 7,5 días


Cal Video para Mike Mansfield Enterprises

Revolución visual
Desde el *Goodbye yellow brick road* (Adiós, camino de ladrillos amarillos), de Elton John, hasta los anuncios de Gillette, los gráficos generados por ordenador están revolucionando la producción de videos y películas. Se pueden crear fácilmente imágenes como éstas aplicando la moderna tecnología del miniordenador y el micro. A una escala mayor, se empleó un Cray-1 para generar más de 20 minutos de metraje para la película *The last starfighter* (El último guerrero del espacio), efectuando cálculos que hubieran tenido ocupado a un micro de 8 bits ¡durante más de 15 años!


Cal Video para B. & D. O.

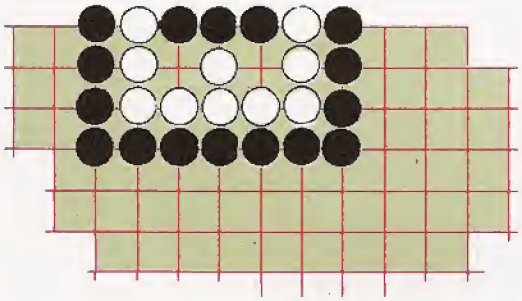
¡No pases de largo!

Prosiguiendo con nuestro proyecto de programación, listaremos las rutinas de entrada/salida para el Commodore 64, el Spectrum y el Amstrad CPC 464/664

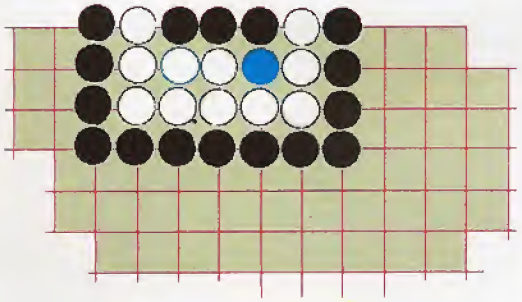
Estas rutinas de entrada/salida tienen por finalidad crear la visualización del tablero para el juego; proporcionan, además, subrutinas de propósito general para imprimir mensajes y aceptar entradas.

En la medida de lo posible, los listados que ofrecemos constituyen el segmento de programa equivalente al que ofrecíamos para el BBC Micro, y gran parte de la descripción ofrecida para los listados del capítulo anterior es igualmente válida para las versiones del *go* para el Commodore 64, el Sinclair Spectrum y el Amstrad CPC 464/664. Para mantenernos en un nivel de sencillez, en todas las versiones del programa se utilizan los mismos números de línea; los algoritmos originales sólo se alteran allí donde las diferencias en cuanto a arquitectura de la máquina y versión de BASIC se vuelven cruciales. Las principales diferencias entre la versión para el BBC Micro y las que ofrecemos aquí, aparte de la forma en que las cuatro máquinas manipulan sus visualizaciones en pantalla, es que las versiones de BASIC del Amstrad, el Spectrum y el Commodore 64 carecen de procedimientos y funciones multilíneas. No obstante, hemos simulado estos últimos mediante el empleo de subrutinas.

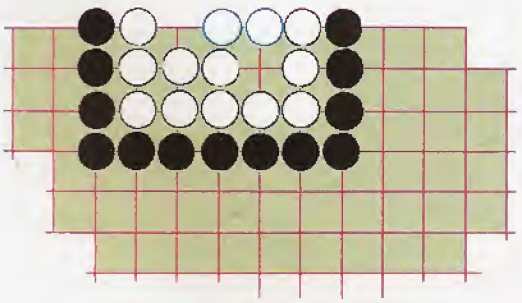
SEKI



¿Quién tiene rodeado a quién?
El grupo blanco está completamente rodeado por el grupo negro, más numeroso, y sólo le quedan dos licencias. Sin embargo, este grupo también tiene rodeado a un grupo negro más pequeño de tres fichas



Error fatal
Si las negras juegan en una de las dos licencias restantes de las blancas, en un intento por capturar el grupo blanco, entonces las blancas pueden jugar en la licencia restante. Puesto que estas dos licencias las comparten las blancas y el grupo negro más pequeño, entonces se podrá suprimir del tablero este grupo negro



Seguridad
Ahora el grupo blanco se puede poner a salvo formando dos ojos, jugando en una de las dos posiciones que se indican. Las negras habrán dejado vacantes las dos licencias compartidas originales, situación que en *go* se denomina SEKI

Caroline Clayton

Módulo Uno

Commodore 64:

```
 10 POKE 56578,PEEK(56578)OR 3:POKE
    56576,(PEEK(56576)AND 252)OR 1
 15 POKE 648,132:POKE 52,128:POKE 56,128
 20 DIM SK%(300)
 30 POKE 53280,0:POKE
    53281,0:PRINT"[BORR.][ABAJ][ABAJ][ABAJ][ABAJ]
    [ABAJ][ABAJ][ABAJ][ABAJ][ABAJ][ABAJ]          [BLANC]
    POR FAVOR ESPERE":GOSUB 170
 40 GOSUB 1270
 50 PRINT"[BORR.]":GOSUB 1730
 60 MOVIMIENTO%=MOVIMIENTO%+1:REM AQUI
    MOVIMIENTOS BLANCAS
 70 IF FIN% GOTO 100
 80 MOVIMIENTO%=MOVIMIENTO%+1:REM AQUI
    MOVIMIENTOS NEGRAS
 90 IF NOT FIN% GOTO 60
100 IP%=22:IM%=9:IW%=1:GOSUB 1990
110 IF A$="S" GOTO 40
115 IF A$<>"N" GOTO100
120 MP%=24:MM%=5:O$="":GOSUB 2160
130 GOTO 130
170 REM RUTINA INICIALIZAR
190 NEGRAS%=1:BLANCAS%=2:COLOR%=3
200 MARCADOR%=4:LICENCIA%=8
220 TABLERO=49152
240 DIM CAPTURA%(2)
250 EJE$="[C 3] ABCDEFGHIJKLMNO"
260 DIM DIR%(4)
270 FOR L=1 TO 4
280 READ DIR%(L)
290 NEXT
300 DATA 16,1,-16,-1
310 GOSUB 390
350 RETURN
390 REM RUTINA LEER MENSAJES
420 DIM MENS$(9)
430 FOR M=0 TO 9
440 READ MENS$(M)
450 NEXT
460 DATA "O.K. ESTOY PENSANDO..."
470 DATA "ENTRADA ILEGAL: "
480 DATA "FICHA YA EN ESE LUGAR: "
490 DATA "ILEGAL. KO EN ESE LUGAR: "
500 DATA "SUICIDIO ILEGAL EN ESE LUGAR: "
510 DATA "O.K. JUEGO TERMINADO. PULSE
    RUN/STOP"
520 DATA ""
```

```
530 DATA "CUANTAS FICHAS DE HANDICAP
    PUEDO          TENER (2-9)?"
540 DATA "DIGITE SU MOVIMIENTO (P.EJ. H8), PASO
    O          ABANDONO:"
550 DATA "QUIERE JUGAR OTRA VEZ (S/N)?"
560 RETURN
1270 REM RUTINA DE INTRODUCCION
1280 GOSUB 1340
1290 GOSUB 1450
1340 REM RUTINA INIC-JUEGO
1345 PILA%=1
1360 A1$="          ":A2$="          "
1370 POSIC%=0:MOVIMIENTO%=1
1380 FIN%=0
1390 CAPTURA%(1)=0:CAPTURA%(2)=0
1410 RETURN
1450 REM RUTINA PANTALLA-TITULOS
1460 POKE 53280,0:POKE 53281,0
1470 PRINT
     "[CLEAR][DOWN][DOWN][DOWN][DOWN]-
          [c 3][RVSON][c V][c I][c
     C][RVSOFF]  [RVSON][c V][c I][c C]"
1480 PRINT"          [RVSON]  [RVSOFF][c D][c
     I]  [RVSON]  [RVSOFF]  [RVSON]  "
1490 PRINT"          [RVSON]  [RVSOFF]  [RVSON]-
     [RVSOFF]  [RVSON]  [RVSOFF]  [RVSON]  "
1550 PRINT"          [c C][RVSON][c I][RVSOFF][c
     V]  [c C][RVSON][c I][RVSOFF][RVSOFF][c F]"
1510 PRINT"[DOWN][DOWN]          [CYAN] POR MARCUS
     JEFFERY"
1520 PRINT"[DOWN][PURPLE]TU JUGARAS CON LAS FICHAS
     [WHITE]BLANCAS[PURPLE], y"
1530 PRINT"EL ORDENADOR (AL SER MAS DEBIL!)
     JUGARA"
1540 PRINT"CON LAS FICHAS [c 3]ROJAS[PURPLE]CON UN
     HANDICAP"
1550 PRINT"          DE VENTAJA."
1560 IP%=20:IM%=7:IW%=1:GOSUB 1990
1565 HND%=VAL(A$)
1570 IF HND%<2 OR HND%>9 GOTO 1560
1590 RETURN
1730 REM RUTINA IMPRESION-TABLERO
1750 PRINT"[HOME][YELLOW]          FICHAS
     CAPTURADAS POR:";
1760 PRINT"[WHITE]          MOVIMIENTO "
1770 PRINT"[YELLOW]          BLANC=[CYAN]
     ";CAPTURA%(2);
1780 PRINT TAB(18);"[YELLOW] NEGRAS=[CYAN]
     ";CAPTURA%(1);
1790 PRINT TAB(31);"[WHITE]";MOVIMIENTO%
1880 PRINT TAB(12);"[DOWN]";EJE$
1810 FOR Y=15 TO 1 STEP -1
1820 PRINT TAB(9);"[c 3]";RIGHT$("  "+STR$(Y),2);"  ";
1830 FOR X=1 TO 15
1840 PP%=PEEK(TABLERO+16*Y+X)
1850 IF PP%=1 THEN PRINT"[c 3][s Q]";:GOTO 1880
1860 IF PP%=2 THEN PRINT"[WHITE][s Q]";:GOTO
     1880
1870 PRINT"[c 6][s +]";
1880 NEXT
1890 PRINT"[c 3]";RIGHT$("  "+STR$(Y),2)
1900 NEXT
1910 PRINT TAB(12);EJE$
1920 PRINT"[DOWN][YELLOW]          ULT. MOV.";
1930 ITCP%=POSIC%:GOSUB 2260:PRINT TAB(24);
     A1$
1940 PRINT TAB(18);"[DOWN][CYAN]";A2$
1950 RETURN
1990 REM RUTINA DE ENTRADA
2000 IS%=0
2010 POLE 780,0:POKE 781,IP%+1:POKE 782,39:SYS
     65520
2015 FOR II=1 TO 79:PRINT CHR$(20);:NEXT
2020 A$=""
2030 PRINT"[WHITE]          ";MENS$(IM%);"  ";
2060 GET I$:IF I$="" GOTO 2060
2065 IF ASC(I$)=13 GOTO 2120
2070 IF ASC(I$)<>20 GOTO 2100
2080 IF IS%>0 THEN IS%=IS%-1:A$=LEFT$(A$,IS%):
     PRINT I$;:GOTO 2060
2085 GOTO 2060
2100 IF IS%<IW% THEN IS%=IS%+1:A$=A$+I$:PRINT
     "[CYAN]";I$;
2110 GOTO 2060
2120 RETURN
2160 REM RUTINA DE MENSAJES
2190 POKE 780,0:POKE 781,MP%:POKE 782,39:SYS
     65520
2195 FOR ML=1 TO 39:PRINT CHR$(20);:NEXT
2200 PRINT"          [RVSON][c
     3]";MENS$(MM%);O$;"[RVSOFF]";
2220 RETURN
2260 REM RUTINA INT-TO-CHAR
2270 IF ITCP%=0 THEN PRINT"[CYAN] HANDICAP";:
     RETURN
2275 C$=CHR$(ITCP%-16*INT(ITCP%/16)+64)+MID$
     (STR$(INT(ITCP%/16)),2)
2280 PRINT"[CYAN]";C$;"          ";:RETURN
```

Amstrad CPC 464/664:

```
 10 MEMORY &9FFF
 20 DIM s(300):REM pila
 30 GOSUB 170: REM inic
 40 GOSUB 1270:REM introduccion
 50 CLS:GOSUB 1730:REM imprimir
    tablero
 60 mov%=mov%+1:REM aqui movimientos
    blancas
 70 IF mas% THEN 60
 80 mov%=mov%+1:REM aqui movimientos
    negras
 90 IF NOT mas% THEN 60
100 ip%=23: im%=9: iw%=1:GOSUB
    1990
110 IF a$="S" THEN 40
115 IF a$<>"N" THEN 100
120 mp%=25:mm%=5:o$="":GOSUB
    2160:REM mensaje
130 END
170 REM rutina inic
190 negras%=1:blancas%=2:color%=3
195 INK 1,26:REM blanco
197 INK 2,24:REM amarillo
200 marcador%=4:licencia%=8
220 tablero=&A000
240 DIM captura%(2)
250 eje$="A B C D E F G H I J K L M N O"
260 GOSUB 390:REM leer mensajes
290 DIM dir%(4)
300 RESTORE 340
310 FOR I%=1 TO 4
320 READ dir%(I%)
330 NEXT I%
340 DATA 16,1,-16,-1
350 RETURN
390 REM rutina leer mensajes
410 RESTORE 460
420 DIM mens$(9)
430 FOR m%=0 TO 9
440 READ mens$(m%)
450 NEXT m%
460 DATA "O.K. ESTOY PENSANDO..."
470 DATA "Entrada ilegal: "
480 DATA "Ficha ya en ese lugar: "
490 DATA "Ilegal. Ko en ese lugar:"
500 DATA "Ilegal. Suicidio en ese lugar: "
510 DATA "      O.K. JUEGO
    TERMINADO"
520 DATA ""
530 DATA "Cuantas fichas puedo tener
    de      handicap (2-9)?"
540 DATA "Digita tu movimiento (p.ej. H8) PASO
    o ABANDONO:"
550 DATA "Quieres jugar otra vez (S/N)?"
560 RETURN
1270 REM rutina de presentacion
1280 GOSUB 1340:REM inic juego
1290 GOSUB 1450:REM pantalla titulos
1300 RETURN
1340 REM rutina inic juego
1345 pila%=1
1360 atari1$="          ":atari2$=""
1370 posicion%=:mov%=1
1380 mas%=0
1390 captura%(1)=0:captura%(2)=0
1410 RETURN
1450 REM rutina pantalla titulos
1460 CLS
1462 FOR c=1 TO 20 STEP 3
1465 RESTORE 1474
1470 MOVE 260+c,320+c
1472 FOR i=1 TO 8:READ x,y:DRAWR x,y,3:
     NEXTi
1474 DATA -10,10,-50,0,-10,-10,0,-80,10,
     -10,50,0,10,10,0,20
1476 MOVER -10,0:DRAWR 20,0
1478 MOVE 380+c,320+c
1480 FOR i=1 TO 8: READ x,y=DRAWR x,y:
     NEXTi
1482 DATA -10,10,-50,0,-10,-10,0,-80,10,
     -10,50,0,10,10,0,80
1484 NEXT c
1500 LOCATE 12,13:PEN  3:PRINT
1510 LOCATE 1,16:PEN 2:PRINT"Jugaras con las
     fichas";
1520 PEN 1:PRINT"blancas";:PEN 2
1530 PRINT"y el ordenador (al ser mas debil!)
     jugara"
1540 PRINT"con las";:PEN 3:PRINT"rojas";:PEN
     2:PRINT"con una ventaja de"
```

```
1550 PRINT TAB(13)"handicap"
1560 ip%=22:im%=7:iw%=1:GOSUB 1990:REM
     entrada
1565 hand%=ASC(a$)-48
1570 IF hand%<2 OR hand%>9 THEN
     1560
1590 RETURN
1730 REM rutina impresion tablero
1735 INK 0,4:BORDER 9:REM magenta/
     verde
1750 LOCATE 3,1:PEN 2:PRINT" Fichas capturadas
     por:";TAB(32)"Movimiento"
1770 PRINT TAB(3)"Blancas =";:PEN 3:PRINT
     captura%(2);
1780 PEN 2:PRINT "Negras =";:PEN 3:PRINT
     captura%(1);
1790 PRINT TAB(33)mov%
1800 PRINT:PRINT TAB(5)eje$
1810 FOR y%=15 TO 1 STEP -1
1820 LOCATE 2,20,-y%:PRINT
     RIGHT$(" "+STR$(y%),2);
1830 FOR x%=1 TO 15
1835 PRINT " ";
1840 p%=PEEK(tablero+16*y%+x%)
1850 IF p%=1 THEN PEN 3:PRINT"O";:GOTO
     1880
1860 IF p%=2 THEN PEN 1:PRINT"O";:GOTO
     1880
1870 PEN 2:PRINT"+";
1880 NEXT x%
1890 PEN 3:PRINT TAB (34)y%
1900 NEXT y%
1910 PRINT TAB(5)eje$
1920 PEN 2:PRINT TAB(3)"Mi ultimo movimiento
     fue:";
1930 itcp%=posicion%:GOSUB 2260:PEN
     1:PRINT TAB(30) atari1$
     1940 LOCATE 16,23:PRINT atari2$
1950 RETURN
1990 REM rutina de entrada
2000 is%=0
2010 LOCATE 3,ip%:PRINT SPACE$(62)
2020 a$=""
     LOCATE 3,ip%:PEN 3:PRINT
     mens$(im%);
2060 i$="":WHILE i$="":i$=INKEY$:
     WEND
2065 IF i$=CHR$(13) THEN 2120
2070 IF i$<>CHR$(127) THEN
     2090
2080 IF is$>0 THEN is%=is%-1:a$=
     LEFT$ (a$,is%):PRINT CHR$(8);" ";
     CHR$(8);
2085 GOTO 2060
2090 IF i$>="a" AND i$<="z" THEN
     i$=CHR$(ASC(i$)-32)
2100 IF is%<iw% THEN is%=is%+1:a$=
     a$+i$:PEN 2:PRINT i$;
2110 GOTO 2060
2120 RETURN
2160 REM rutina de mensajes
2190 LOCATE 3,mp%:PRINT SPACE$
     (36);
2200 LOCATE 3,mp%:PEN 3:PRINT mens$(mm%);
     o$
2220 RETURN
2260 REM rutina int-to-char
2270 IF itcp%=0 THEN PEN 1:PRINT"Handicap"
     ;:RETURN
2275 c$=CHR$(itcp%-16*INT(itcp%/16)+64)
     :r$=MID$(STR$(INT(itcp%/16)),2)
2280 PEN 1:PRINT c$;r$;:RETURN
```

Sinclair Spectrum

```
 10 CLEAR 63999
 20 DIM s(300)
 30 PRINT AT 10,10;"POR FAVOR ESPERA":GO
    SUB 170
 40 GO SUB 1270
 50 CLS:GO SUB 1730
 60 LET movimiento=movimiento+1:REM aqui
    movimientos blancas
 70 IF fin THEN GO TO 100
 80 LET movimiento=movimiento+1:REM aqui
    movimientos negras
 90 IF NOT fin THEN GO TO 60
100 LET ip=20: LET im=10: LET iw=1: GO SUB
    1990
110 IF a$="S" THEN GOTO 40
115 IF a$<> "N" THEN GO TO 100
120 LET mp=21: LET mm=6: LET o$="": GO
    SUB 2160
130 STOP
170 REM rutina inicializar
190 LET negras=1: LET blancas=2: LET
    color=3
200 LET marcador=4: LET licencia
    =8
220 LET tablero=64000
240 DIM c(2)
260 GO SUB 390
290 DIM d(4)
300 RESTORE 340
310 FOR I=1 TO 4
320 READ d(I)
330 NEXT I
340 DATA 16,1,-16,-1
350 RETURN
390 REM rutina leer-mensajes
410 RESTORE 460
420 DIM m$(10,43)
430 FOR m=1 TO 10
440 READ m$(m)
450 NEXT m
460 DATA "O.K. ESTOY PENSANDO..."
470 DATA "Entrada ilegal: "
480 DATA "Ficha ya en ese lugar: "
490 DATA "Ilegal. Ko en ese lugar: "
500 DATA "Ilegal. Suicidio en ese
    lugar:"
510 DATA "      O.K. JUEGO
    TERMINADO."
520 DATA ""
530 DATA "Cuantas fichas puedo tener de
    handicap (2-9)?"
540 DATA "Digita tu movimiento (p.eje. H8),
    PASO o ABANDONO :"
550 DATA "Quieres jugar otra vez      (S/N)?"
560 RETURN
1260 :
1270 REM rutina de presentacion
1280 GO SUB 1340
1290 GO SUB 1450
1300 RETURN
1340 REM rutina inic-juego
1345 LET pila=1
1360 LET x$="      ": LET y$="      "
1370 LET posicion=0: LET movimiento=1
1380 LET fin=0
1390 LET c(1)=0: LET c(2)=0
1410 RETURN
1450 REM rutina pantalla titulos
1460 BORDER 0: PAPER 0: CLS
1470 INK 2: PRINT AT 3,12; "sh23sh1
     sh23sh1"
1480 PRINT AT 4,12;"sh84sh3 sh8 sh8"
1490 PRINT AT 5,12;"sh8 sh8    sh8 sh8"
1500 PRINT AT 6,12;"132    132"
1510 PRINT AT 8,7; INK 5;"por Marcus
     Jeffery"
1520 PRINT:PRINT INK 3:"Tu jugaras con las
     fichas"; INK 7:"blancas"; INK 3
1530 PRINT INK 3;"y el ordenador (por ser mas
     debil)"
1540 PRINT INK 3;"jugara con las fichas";INK
     2;"rojas  " ;INK 3;"con"
1550 PRINT INK 3;"      una ventaja de
     handicap."
1560 LET ip=15: LET im=8: LET iw=1: GO SUB
     1990
1565 LET hand=CODE (a$)-48
1570 IF hand<2 OR hand>9 THEN GO TO
     1560
1590 RETURN
1730 REM rutina impresion tablero
1750 PRINT AT 0,0; INK 6;" Fichas capturadas
     por:";
1760 PRINT INK 7;"      Mov."
1770 PRINT INK 6;" Blancas=
     "; INK 5;c(2);
1780 PRINT TAB 10; INK 6;"Negras=
     "; INK 5;c(1);
1790 PRINT TAB 27; INK 7; movimiento
1810 FOR y=15 TO 1 STEP -1
1820 PRINT AT 17-y,5; INK 2;(" "+STR$ y)(LEN
     STR$ y TO);" ";
1830 FOR x=1 TO 15
1840 LET pp=PEEK (tablero+16*y+x)
1850 IF pp=1 THEN PRINT PAPER 4; INK
     2;"O";:GO TO 1880
1860 IF pp=2 THEN PRINT PAPER 4; INK
     7;"O";:GO TO 1880
1870 PRINT PAPER 4; INK 0;"+";
1880 NEXT x
1890 PRINT INK 2;" ";y
1900 NEXT y
1910 PRINT TAB 8; INK 2;"ABCDEFGHIJKLMNO"
1920 PRINT INK 6;"Ultimo movimiento: ";
1930 LET itcp=posicion: GO SUB 2260: PRINT
     TAB 20: INK 5;x$
1940 PRINT AT 20,20; INK 5;y$;
     BEEP 1,0
1950 RETURN
1990 REM rutina de entrada
2000 LET is=0
2010 PRINT AT ip,0;:FOR i=1 TO 62: PRINT "
     ";:NEXT i
2020 LET a$=" "
2030 PRINT AT ip,0; INK 7; m$(im);
2060 LET i$=INKEY$: IF i$=" " THEN GO TO
     2060
2065 IF CODE i$=13 THEN GO TO
     2120
2070 IF CODE i$<>12 THEN GO TO
     2090
2080 IF is>0 THEN LET is=is-1:LET a$=a$(1 TO
     LEN a$-1): PRINT CHR$ (8);"
     ";CHR$(8);:GO TO 2060
2085 GO TO 2060
2090 IF i$>="a" AND i$<="z" THEN LET
     i$=CHR$ (CODE i$-32)
2100 IF is<iw THEN LET is=is+1: LET
     a$=a$+i$: PRINT INK5; i$;
2110 GO TO 2060
2120 RETURN
2160 REM rutina de mensajes
2190 PRINT AT mp,0;:FOR m=1 TO 30: PRINT "
     ";: NEXT m
2200 PRINT AT mp,0;  INK 2;m$(mm, TO 26);
     o$;
2220 RETURN
2260 REM rutina int-to-char
2270 IF itcp=0 THEN PRINT INK
     5;"Handicap";:RETURN
2275 LET c$=CHR$ (itcp-16*INT (itcp/16)+64):
     LET r$=STR$ (INT (itcp/16))
2280 PRINT INK 5;c$;r$;:RETURN
```

Datos básicos (II)

Proseguiremos nuestro análisis del mapa de memoria del C64, con información facilitada por Commodore Business Machines

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
LASTPT	0017-0018	23-24	Última dirección de serie temporal
TEMPST	0019-0021	25-33	Pila para series temporales
INDEX	0022-0025	34-37	Área punteros utilidades
RESHO	0026-002A	38-42	Producto de multiplicar coma flotante
TXTTAB	002B-002C	43-44	Puntero: inicio texto en BASIC
VARTAB	002D-002E	45-46	Puntero: inicio variables BASIC
ARYTAB	002F-0030	47-48	Puntero: inicio tablas BASIC
STREND	0031-0032	49-50	Puntero: fin tablas BASIC (+ 1)
FRETOP	0033-0034	51-52	Puntero: extremo inferior almacenamiento series
FRESPC	0035-0036	53-54	Puntero series utilidades
MEMSIZ	0037-0038	55-56	Puntero: dirección más alta empleada por BASIC
CURLIN	0039-003A	57-58	Número línea actual BASIC
OLDLIN	003B-003C	59-60	Número línea anterior BASIC
OLDTXT	003D-003E	61-62	Puntero: sentencia BASIC de CONT
DATLIN	003F-0040	63-64	Número actual línea DATA
DATPTR	0041-0042	65-66	Puntero: dirección ítem actual DATA
INPPTR	0043-0044	67-68	Vector: rutina INPUT
VARNAM	0045-0046	69-70	Nombre actual variable BASIC
VARPNT	0047-0048	71-72	Puntero: datos variables en curso de BASIC
FORPNT	0049-004A	73-74	Puntero: variable índice de FOR/NEXT
	004B-0060	75-96	Puntero temp/Área datos
FACEXP	0061	97	Acumulador # 1 coma flotante: exponente

Ratón mágico

El Magic Mouse, de SMC Supplies, es el primer dispositivo de ratón creado para el Commodore 64

A la vista de la actual popularidad de que gozan los sistemas operativos basados en ratón, quizá resulte difícil creer que inicialmente se produjera tanta controversia dentro de la industria del ordenador en cuanto a si el sistema basado en ratón era o no aceptable, por no hablar de deseable. El fracaso comparativo del Apple Lisa, el primer micro que tuvo un sistema operativo basado en ratón, pareció confirmar la idea de que los usuarios no querían estos sistemas.

El posterior éxito del Macintosh silenció, sin embargo, las críticas, y justificó la fe que tenía depositada Apple en su sistema. Las actitudes han cambiado drásticamente; los fabricantes de ordenadores se están apresurando a producir máquinas basadas en ratón, y muchas empresas independientes están editando paquetes de ratón para máquinas existentes.

El Magic Mouse (ratón mágico) de SMC Supplies es uno de los primeros de tales productos que ha salido para el Commodore 64. Con el ratón se incluye el software, tanto en cassette como en disco, que contiene cuatro paquetes de aplicaciones. El ratón SMC es más grande que la mayoría de los dispositivos similares, y con sus 125 por 66 por 50 mm, su tamaño es casi el doble que el del ratón Apple.

El Magic Mouse también tiene tres botones de colores en la parte delantera, frente a los dos que suele haber normalmente. Los botones se emplean para seleccionar funciones que, por supuesto, varían a tenor de la aplicación que se esté utilizando. Por dentro hay una bola de plástico duro que hace presión contra un par de codificadores de eje.

Aparentemente, el diseño original llevaba una conexión de bola metálica estándar, pero se descubrió que la misma no era adecuada y se retrasó el lanzamiento del dispositivo para permitir la incorporación de los recambios de goma. No obstante, el ratón parece funcionar a la perfección.

Eficaz herramienta
El SMC Magic Mouse es uno de los primeros sistemas basados en ratón que se desarrolla para el Commodore 64. A pesar de no estar diseñado como un auténtico paquete WIMP *(windows, icons, mice, programming:* ventanas, iconos, ratones, programación), se puede emplear eficazmente como una herramienta de desarrollo para programadores. El ratón se enchufa en la puerta para palanca de mando del Commodore 64 y los iconos en pantalla se seleccionan mediante los tres botones existentes en el ratón

Crispin Thomas

Programas de utilidades

El paquete Magic Mouse no es un sistema operativo como el del Apple Macintosh, sino más bien una facilidad para el diseño de gráficos. Los cuatro programas que se entregan con el ratón son *Hi-res designer* (diseñador en alta resolución), *Sprite designer* (diseñador de sprites), *Icon designer* (diseñador de iconos) y *Mouse controller* (controlador del ratón).

Para poder ejecutar correctamente el software, primero se debe cargar (LOAD) el archivo del sistema y calibrar el cursor del ratón. El sistema basado en disco se carga automáticamente con un menú principal que permite seleccionar cualquiera de los cuatro programas de utilidades. El sistema basado en cassette es ligeramente distinto. Aquí el archivo del sistema forma parte del *Hi-res designer* y, por consiguiente, se ha de cargar este programa antes de poder utilizar cualquiera de los otros tres. Asimismo, el cursor debe posicionarse en la esquina inferior derecha de la pantalla antes de comenzar.

Hi-res designer es un programa de «caja de colores» de aspecto muy similar al software Koala Pad. Al cargarlo, la pantalla pasa a una serie de casilleros, cada uno de los cuales contiene una opción diferente, como Recuadro, Dibujar o Rellenar. Desplazando el cursor del ratón hasta uno de estos casilleros y pulsando el botón rojo del ratón, se selecciona dicha opción. De modo similar, para elegir ya sea el color de primer plano o el de fondo, uno simplemente debe llevar el cursor hasta las líneas de los 16 colores disponibles y seleccionar una. Para mover el «lienzo», se pulsa el botón azul.

Los paquetes *Icon* y *Sprite designer* son muy similares y en realidad sólo se diferencian en sus aplicaciones finales. Como la mayoría de los programas de esta naturaleza, el *Sprite designer* presenta una cuadrícula de 24 columnas por 21 filas. Los pixels se encienden pulsando el botón rojo cuando el cursor se halla sobre la posición adecuada. El sprite propiamente dicho aparece en una ventana situada en la esquina superior derecha de la pantalla, y cuando el cursor se sale de la cuadrícula en la esquina inferior derecha aparecen varias opciones. Éstas son para aplicaciones tales como ampliar el sprite, cambiar su color o pasar a un sprite diferente.

El *Icon designer* aparece casi idéntico, con una

cuadrícula en la esquina y el icono, que aparece arriba en una ventana. La diferencia existente entre ambos es que *Icon designer* se utiliza para construir gráficos definidos por el usuario, como, por ejemplo, paisajes, que se pueden guardar y volver a utilizar.

El último de los programas que suministra el software Magic Mouse es el *Mouse controller*. Este programa *no* hace nada por sí mismo, sino más bien proporciona el software activador que permite que sus programas se ejecuten desde el sistema. Aunque a primera vista éste es el menos útil de los cuatro programas, en realidad es el que posee el mayor potencial, puesto que estimulará al usuario a escribir su propio software para el ratón.

Características del cursor

La gran ventaja que tienen los sistemas basados en ratón sobre los digitalizadores es la uniformidad del cursor, que permite el posicionamiento exacto de las líneas y colores que se desea dibujar. Un digitalizador se basa en una cuadrícula de nudos interconectados; sin embargo, su resolución es inferior a la de la pantalla. Por tanto, cuando el «lápiz» se halla entre dos nudos, el ordenador no puede decidir cuál es la posición que se pretende y tiende, entonces, a saltar entre una y otra. Por supuesto, con frecuencia esto suele tener resultados desastrosos para la imagen que uno intenta construir. Cuando se utiliza el ratón, sin embargo, su cursor reacciona sólo a los cambios que se producen en sus potenciómetros, provocados por los movimientos de la bola, de modo que si no se mueve el ratón el cursor permanecerá quieto.

No obstante, el sistema no es perfecto. Algunos aspectos del *Hi-res designer* no son tan amables como los de otros paquetes. Por ejemplo, la instrucción RUB sólo permite borrar errores pixel a pixel. Ello lleva su tiempo y, si se borran pixels equivocados, se corre el riesgo de generar nuevos errores.

Un procedimiento mucho más acertado consiste en borrar todos los añadidos al lienzo desde la última vez que uno pasó por la pantalla de menú. Por supuesto, esto significa que en el proceso se borrará una buena cantidad de trabajo, pero al menos el usuario tendrá la posibilidad de empezar con una hoja en blanco.

El manual del Magic Mouse es extraordinariamente didáctico. Cada uno de los programas posee tanto una explicación completa sobre la forma en que trabaja el software de desarrollo, como información de programación para que usted incorpore el producto acabado a su propio software, incluyendo programas de muestra. Ello revela un buen trabajo de previsión por parte de SMC, a la que hay que felicitar por su exhaustivo y útil manual.

Si bien el Magic Mouse de SMC Supplies dista mucho de ser el Macintosh Emulator, la empresa obviamente ha desarrollado el software inicial como una herramienta para el programador, similar al ratón AMX o al software DR GEM. Sin duda alguna, la empresa espera que los programadores produzcan software adicional que consiguientemente aumente el atractivo del sistema. Puede que este enfoque no produzca ventas rápidas, pero, en cambio, es probable que asegure una larga vida al Magic Mouse.

John Clementson

Desarrollo artístico
Estas imágenes se crearon mediante el programa de aplicaciones *Hi-res designer* para usar con el SMC Magic Mouse. El programa admite las instrucciones usuales de que disponen esta clase de paquetes, incluyendo LINE, DRAW, FILL y CIRCLE. Se pueden utilizar hasta 16 colores diferentes para componer la imagen, y hay una amplia gama de distintos «pinceles», que producen líneas de diverso espesor y textura

MAGIC MOUSE

INTERFACE

Se enchufa en la puerta para palanca de mando del Commodore 64

SOFTWARE

Se entrega con cuatro programas de aplicaciones tanto en disco como en cassette

DOCUMENTACION

El manual ofrece detalles completos sobre cómo instalar el ratón, cómo usar el software existente y cómo incorporar los resultados en los propios programas del usuario

VENTAJAS

Parece bien construido y fiable, y los usuarios tendrán pocos problemas para transferir a sus propios programas los sprites y gráficos definidos por el usuario

DESVENTAJAS

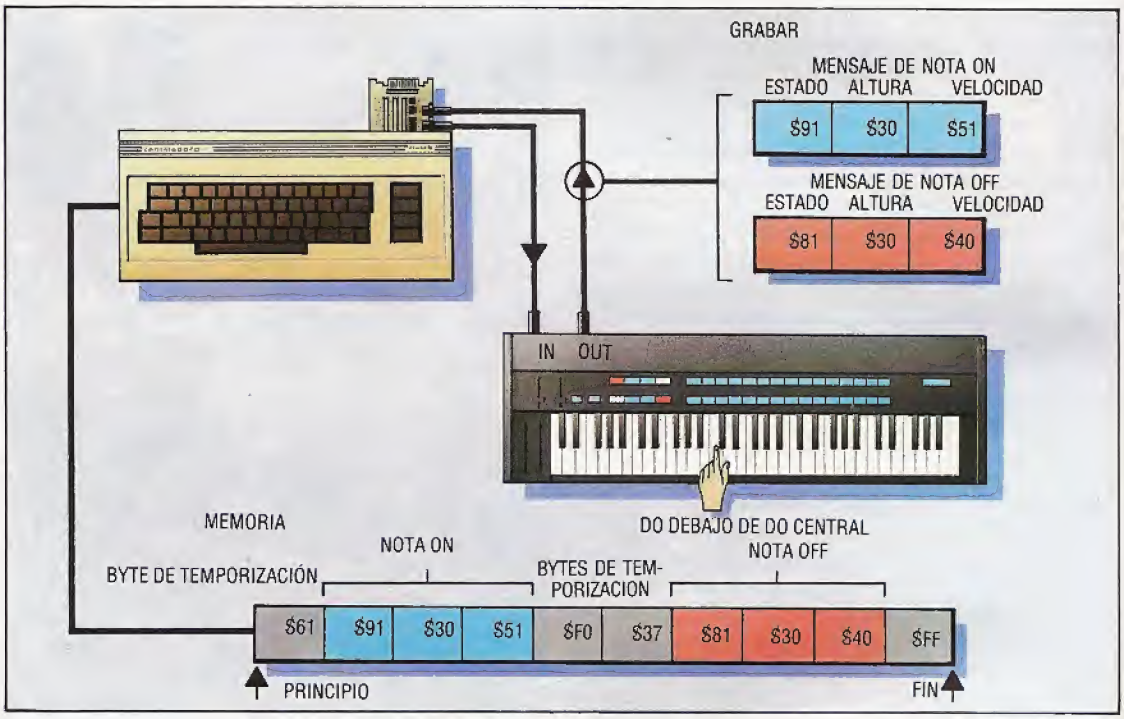
El paquete en conjunto carece del alcance de muchos otros programas para ratón, y se encontrará muy retrasado respecto a los paquetes más sofisticados

Alta fidelidad

Escribiremos un programa que permitirá que el ordenador actúe como un dispositivo digital grabador y reproductor en tiempo real

Cuando se pulsan las teclas de un teclado MIDI, o cuando se activa un dispositivo controlador como un mando de control de altura, estos «eventos» se transmiten en forma de bytes de datos al conector MIDI OUT de la parte trasera del sintetizador. Gran parte de nuestro capítulo anterior lo dedicamos a ofrecer detalles acerca del significado de estos valores de bytes, para que usted pudiera escribir programas que aceptaran datos MIDI y los procesara. El ordenador también se puede utilizar como una fuente de datos MIDI, que se pueden transmitir a un sintetizador con el objeto de activar sus circuitos de generación de sonido. Un ejemplo obvio de este segundo uso es el de organizar el ordenador como un secuenciador, permitiendo entrar y editar en *tiempo de pasos* (programación de intervalos individuales dentro de una barra) frases musicales desde el teclado del ordenador. Ello se hace antes de la transmisión a través de la MIDI para activar el sintetizador que tocará la melodía. De hecho, es una labor relativamente simple: la MIDI es capaz de mucho más. ¡Piense en un secuenciador que utilice la totalidad de los 16 canales de la MIDI para controlar una matriz de diferentes teclados y una caja de ritmos!

Podemos combinar varios aspectos de la forma en que se puede utilizar un ordenador para que interactúe con instrumentos MIDI, asumiendo la tarea relativamente directa de diseñar un trozo de software que pueda grabar en la memoria del ordenador una melodía tocada en tiempo real. Luego, en respuesta a una instrucción, el programa reproducirá la melodía desde la MIDI nuevamente hacia el sintetizador. Esto, por supuesto, es una grabación digital.



La voz de su micro
Nuestro diagrama ilustra cómo se grabaría una sola nota utilizando el programa *Grabación digital*. Durante la fase de grabación, el sintetizador envía un mensaje de *nota on* cuando se pulsa la tecla, y un mensaje de *nota off* cuando la misma se libera. Estos mensajes se almacenan en la memoria, junto con información de temporización que registra cuántos períodos de 2 milisegundos transcurren entre un evento y el siguiente. La información de temporización entre los mensajes de *nota on* y *nota off* en el ejemplo se retiene en dos bytes, mostrando cómo se maneja el desbordamiento del temporizador. Durante la reproducción, el programa utiliza la información de temporización para demorar la transmisión de los mensajes de nota, de modo que se reproduzca la misma «articulación»

A pesar de que superficialmente es similar a la grabación analógica utilizando una grabadora de cintas y una cinta magnética, la grabación digital a través de la MIDI es radicalmente diferente. Cuando se efectúa una grabación en un trozo de cinta, los verdaderos sonidos musicales se codifican en patrones magnéticos en la superficie de la cinta, y la posición de estos patrones en la cinta determina el orden por el cual se han de reproducir los sonidos durante la reproducción, así como los intervalos de tiempo entre un sonido y el siguiente. Imaginemos que se están grabando dos notas de esta forma, suponiendo que entre éstas había un vacío de dos segundos. En este intervalo la cinta continuaría pasando las cabezas de cinta como si no se hubiera grabado nada. Cuando se reprodujera la cinta, la longitud de esta sección de cinta en blanco determinaría el intervalo de tiempo entre que el oyente escuchara la primera y la segunda nota.

En un sistema de grabación digital que utilice MIDI, las dos notas corresponderían a dos eventos de pulsaciones de teclas, cada uno de ellos generando un mensaje MIDI. El problema de una grabación en tiempo real es que debemos tener alguna forma de grabar el intervalo de tiempo entre que el

ordenador recibe el primero y el segundo mensaje MIDI. Utilizando la analogía de la grabación en cinta, la forma más obvia de hacerlo sería almacenando los mensajes en una matriz de la memoria, almacenando los valores de bytes «en blanco» para cada unidad de tiempo en el intervalo. Por tanto, durante la reproducción, las dos notas se tocarían con el intervalo de tiempo correcto. No es difícil comprender que este método de grabar eventos MIDI en tiempo real supone potencialmente un gran derroche de preciosa memoria.

Una alternativa para esta analogía simplista de una grabadora de cinta analógica consiste en grabar la información de tiempo en forma de bytes en lugar de en forma de valores de bytes «en blanco». Por ejemplo, si hay 50 unidades de tiempo entre la primera y la segunda nota, en la matriz de la memoria se entraría un byte de temporización que contuviera el valor 50, en vez de 50 bytes «en blanco».

Ello supone un considerable ahorro de memoria, pero también dificulta la predicción de la longitud máxima de una melodía que se puede grabar en una cantidad de memoria dada. Las melodías que implican un rápido trabajo de los dedos, por ejemplo, requerirán más bytes para grabar la gran cantidad de mensajes de notas *on/off* transmitidos por el sintetizador que una pieza más lenta de la misma longitud. La utilización del mando de altura también tiende a devorar enormes cantidades de memoria, debido a la cantidad de mensajes MIDI que genera su operación.

Grabación digital

En el programa que ofrecemos necesitaremos, en consecuencia, alguna clase de contador para grabar los intervalos de tiempo entre los eventos. El contador que utilizamos es una variable llamada CLOCKS (relojes). La velocidad a la cual se incrementa este contador establece la precisión con la que se puede determinar la temporización y, por consiguiente, la fidelidad de la grabación. Al seleccionar el intervalo de tiempo óptimo hemos debido hacer un compromiso, optando por un intervalo de alrededor de dos milisegundos. Éste no es tan pequeño como para ser menor que los tiempos de ejecución de las rutinas, pero tampoco es tan grande como para incidir negativamente en la precisión de la grabación. Este intervalo de tiempo se denomina la *resolución* de la grabadora, dado que es el menor intervalo de tiempo que se puede grabar. El intervalo lo genera un temporizador de uno de los chips CIA o VIA del micro.

El formato del mensaje MIDI se puede ampliar ligeramente para tener en cuenta la temporización. Esto se hace colocando antes de cada mensaje MIDI un único byte que representa la cantidad de intervalos de tiempo que han transcurrido desde el último mensaje. Este único byte puede registrar hasta 239 unidades de tiempo y, por tanto, puede tener valores de entre $0 y $EF.

Si pasan 240 intervalos sin que se reciba un mensaje, se graba un mensaje de desbordamiento. Este mensaje es el byte $F0. Por último, se ha elegido un mensaje de un único byte, $FF, para señalar el final de la secuencia. Los valores de byte de $F1 a $FE están reservados para sus propios punteros de programa.

Marcus Wilson-Smith

Operación del programa
La operación del programa *Grabación digital* para el Commodore 64 es directa. La selección de la opción «grabar» grabará los eventos que lleguen a MIDI IN. La grabación se detiene ya sea cuando no se utiliza la memoria disponible o bien cuando se pulsa la barra espaciadora. La opción «reproducir» reproducirá luego la secuencia de eventos grabada en MIDI OUT. La reproducción continua hasta que se pulsa la barra espaciadora, o bien al llegar al fin de la secuencia. Toda ulterior grabación se sobreescribirá en la anterior

Programa «Grabación digital»

Cargador de BASIC

```
 10 FOR I = 49152 TO 49581
 20 READ X: POKE I,X: S=S+X:NEXT
 30 READ X: IF S<>X THEN PRINT"OK SYS49152 TO RUN":END
 40 PRINT "ERROR EN SUMA DE CONTROL"
 50 END
100 DATA 76,7,192,2,83,201,241,169,3
110 DATA 141,0,222,169,21,141,0,222,169
120 DATA 255,141,173,193,169,0,141,6
130 DATA 220,169,8,141,7,220,169,17,141
140 DATA 15,220,32,228,255,201,0,240
150 DATA 249,201,69,208,1,96,201,82,240
160 DATA 4,201,80,208,236,72,120,169
170 DATA 127,141,0,220,169,173,133,247
180 DATA 169,193,133,248,169,173,141,5
190 DATA 192,169,241,141,6,192,160,0
200 DATA 104,201,82,8,240,66,169,2,32
210 DATA 69,193,32,44,193,201,255,208,4
220 DATA 40,76,6,193,141,4,192,201,240
230 DATA 173,4,192,240,12,32,246,192
240 DATA 240,251,206,4,192,208,246,176
250 DATA 223,32,44,193,72,173,0,222,41
260 DATA 2,240,249,104,141,1,222,16,3
270 DATA 32,53,193,202,208,233,174,3
280 DATA 192,16,195,169,1,32,69,193,140
290 DATA 4,192,162,255,32,246,192,240
300 DATA 16,238,4,192,173,4,192,201,240
310 DATA 144,6,32,21,193,140,4,192,173
320 DATA 0,222,41,1,208,6,224,1,48,224
330 DATA 16,243,173,1,222,48,11,224,0
340 DATA 48,213,208,27,174,3,192,16,11
350 DATA 201,248,176,223,201,240,176
360 DATA 196,32,56,193,72,173,4,192,32
370 DATA 21,193,140,4,192,104,32,21,193
380 DATA 202,240,178,208,197,173,1,220
390 DATA 73,239,208,18,104,104,40,208,4
400 DATA 169,255,145,247,88,169,0,32,69
410 DATA 193,76,37,192,173,13,220,41,2
420 DATA 96,145,247,238,5,192,208,18
430 DATA 238,6,192,208,13,104,104,104
440 DATA 169,3,32,69,193,76,6,193,177
450 DATA 247,230,247,208,2,230,248,96
460 DATA 162,2,201,192,144,5,201,224
470 DATA 176,1,202,142,3,192,232,96,10
480 DATA 170,189,96,193,133,249,189,97
490 DATA 193,133,250,160,0,177,249,240
500 DATA 6,32,210,255,200,208,246,160,0
510 DATA 96,104,193,134,193,146,193,158
520 DATA 193,82,32,61,32,82,69,67,79,89
530 DATA 68,44,80,32,61,32,80,76,65,89
540 DATA 44,69,32,61,32,69,88,73,84,13
550 DATA 0,147,82,69,67,79,82,68,73,78
560 DATA 71,13,0,147,80,76,65,89,66,65
570 DATA 67,75,32,13,0,79,85,84,32,79
580 DATA 70,32,77,69,77,79,82,89,13,0
590 DATA 240
600 DATA 52178:REM"SUMA DE CONTROL"
```

Listado assembly

```
* = $C000
GETIN     = $FFE4          ;ENTRAR UN CARACTER
CHROUT    = $FFD2          ;SACAR UN CARACTER
          JMP  START
NOBYTS    *=*+1            ;NUM. DE BYTES EN MENSAJE ACTUAL
CLOCKS    *=*+1            ;NUM. PERIODOS 2MSEG DESDE ULTIMO
                           MENSAJE
FREMEN    *=*+2            ;NUM. DE BYTES LIBRES
DATREG    = $DE01          ;REG TRANSMISION/RECEPCION DEL ACIA
STREG     = $DE00          ;REG ESTADO/CONTROL PARA EL ACIA
STROPKY   = $ER
MEM       = $F7            ;PUNTERO MEMORIA DATOS
PTR       = $F9            ;PUNTERO LECTURA SERIES
START     =*
;++++ RUTINAS PREPARACION ++++
          LDA  # $03
          STA  STREG       ;INICIALIZAR 6850
          LDA  # $16
          STA  STREG
          G10       LDA # $FF
```

```
STA     LOWMEM          ;INIC VACIA
        LDA  # $00
        STA  # DC06
        LDA  # $08
        STA  $DC07
        LDA  # $11
        STA  $DC0F      ;EMPEZAR TEMPORIZADOR B
G20         JSR GETIN
        CMP  # $00
        BEQ  G20
        CMP  # 69        ;PULSAR "E" PARA SALIR
        BNE  G22
        RTS
G22     = *             ;NO SALIR
        CMP  # 82
        BEQ  G30
        CMP  # 80
        BNE  G20
                        ;AQUI COMIENZA PROG EN TIEMPO REAL
G30         PHA         ;INSTRUCCION SALVAR GRABAR/PB
        SEI
        LDA  # $7F
        STA  $DC00      ;EXPLORAR ULTIMA FILA DE TECLAS
        LDA  # < LOWMEM
        STA  MEM
        LDA  # > LOWMEM
        STA  MEM + 1
        LDA  # < FRBYTS
        STA  FREMEM
        LDA  # > FRBYTS
STA     FREMEM + 1
        LDY  # $00
        PLA
        CMP  # 82
        PHP             ;GUARDAR FLAG Z = GRABAR
        BEQ  R00

P00     = *             ;RUTINA REPRODUCCION

        LDA  # $02
        JSR  STROUT
P05     JSR READ
        CMP  # $FF
        BNE  P07        ;SALIR SI SE HA LLEGADO AL FINAL
        PLP             ;AJUSTAR PILA
        JMP  C20        ;PROCESO DE SALIDA
P07     STA CLOCKS      ; ARRASTRE = LECTURA $F0
        CMP  # $F0
        LDA  CLOCKS
        BEQ  P30        ;JMP SI NO HAY TIEMPO PARA ESPERAR
P10     JSR CHECH
        BEQ  P10
        DEC  CLOCKS
        BNE  P10        ;JMP SI TODAVIA HAY TIEMPO
        BCS  P05        ;LEER SIGUIENTE RELOJ SI $F0

P30     = *             ;LEER BYTE SIGUIENTE MENSAJE
JSR     READ
PHA                     ;GUARDAR BYTES DATOS

P35     = *             ;ESPERAR REGISTRO TX
LDA     STREG
AND     # $02
BEQ     P35             ;JMP SI AUN LLENO

PLA                     ;TOMAR DATOS
STA     DATREG          ;ENVIAR A MIDI
BPL     P50             ;JMP SI NO BYTE ESTADO
JSR     GETNO           ;TOMAR NUM DE BYTES DE DATOS
P50     DEX             ;DEC. CONTADOR BYTES
BNE     P30             ;TOMAR SIGUIENTE BYTES DATOS
LDX     NOBYTS
BPL     P05             ;TOMAR BYTE RELOJ

R00     = *             ;RUTINA GRABACION

LDA     # $01
JSR     STROUT
STY     CLOCKS          ;CLOCKS = 0
R05     LDX  # $FF      ;INIC A NO DATOS
R10     JSR  CHECK
BEQ     R20
INC     CLOCKS
LDA     CLOCKS
CMP     # $F0
BCC     R20
JSR     STORE
STY     CLOCKS

R20     = *             ;COMPROBAR DATOS MIDI
LDA     STREG
AND     # $01
BNE     R40             ;JMP SI DATOS
CPX     # $01
BMI     R10             ;JMP SI ESPERANDO
BPL     R20             ;JMP SI MENSAJE-MID

R40     = *             ;TOMAR LOS DATOS
LDA     DATREG
BMI     R50             ;JMP SI BYTE ESTADO
CPX     # $00
BMI     R10             ;JMP SI NINGUN DATO CANAL
BNE     R80             ;JMP SI MENSAJE-MID
LDX     NOBYTS          ;RESTABLECER CONTADOR BYTES
BRL     R60

R50     = *             ;PROCESO BYTE ESTADO
CMP     # $F8
BCS     R20             ;IGNORAR SI SYS REALTIME
CMP     # $F0
BCS     R05             ;JMP SI NO EST. CANAL
JSR     GETNO           ;TOMAR NUM. BYTES DATOS
```

```
R60     = *             ;GRABAR LOS DATOS
PHA
LDA     CLOCKS
JSR     STORE
STY     CLOCKS          ;RESTABLECER CONTADOR
PLA

R80     = *             ;GRABAR DATOS MIDI
JSR     STORE
DEX
BEQ     R10
BNE     R20

CHECK   = *
LDA     $DC01
EOR     # STOPKY       ;NO AFECTA AL ARRASTRE
BNE     C40
PLA
PLA                     ;AJUSTAR PILA
PLP                     ;COMPROBAR REC/PB
BNE     C20             ;JMP SI PLAYBACK
LDA     # $FF
STA     (MEM),Y
C20     = *
CLI
LDA     # $00
JSR     STROUT
JMP     G20             ;ESPERAR SIGUIENTE INSTRUCCION

C40     = *             ;COMPROBAR TEMPORIZADOR B
LDA     $DC0D
AND     # $02          ;FLAG Z = TIEMPO NO AGOTADO
RTS

STORE   = *             ;ALMACENAR DATOS EN MEMORIA

STA     (MEM),Y
INC     FREMEM          ;COMPROBAR MEMORIA LIBRE
BNE     POINT
INC     FREMEM + 1
BNE     POINT
PLA
PLA
PLA
LDA     # $03          ;MENSAJE DE ERROR
JSR     STROUT
JMP     C20             ;RESTAURAR PROGRAMA

READ    = *             ;LEER DATOS DE LA MEMORIA

LDA     (MEM),Y

POINT   = *             ;ACTUALIZAR PUNTERO MEMORIA

INC     MEM
BNE     P20
INC     MEM + 1
P20     RTS

GETNO   = *             ;TOMAR NUM. BYTES DE DATOS
                PARA ESTADO EN ACC
LDX     # $02          ;INIC A 2 BYTES DATOS
CMP     # $C0
BCC     N10
CMP     # $E0
BCS     N10
DEX                     ;1 BYTE DATOS SI C0<=A<E0
N10     STX  NOBYTS
INX                     ;SUMAR 1 PARA ESTADO
RTS

STROUT  = *             ;IMPRIMIR SERIE

ASL     A
TAX
LDA     MESTAB,X
STA     PTR
LDA     MESTAB+1,X
STA     PTR+1
LDY     # $00
M10     LDA  (PTR),Y
BEQ     M70
JSR     CHROUT
INY
BNE     M10
M70     LDY  #$00
RTS

MESTAB  = *             ;TABLA MENSAJES

.WORD   MESS0
.WORD   MESS1
.WORD   MESS2
.WORD   MESS3

MESS0   .BYTE 'R = GRABAR, P = TOCAR, E = SALIR',$0D,0


MESS1   .BYTE 147,'GRABANDO',$0D,0


MESS2   .BYTE 147,'PLAYBACK',$0D,0


MESS3   .BYTE 'FUERA DE MEMORIA',$0D,0


LOWMEM  = *             ;COMIENZO DE MEMORIA DATOS
FRBYTS  = LOWMEM - $D000
```

Lista activa

Concluiremos nuestra serie dedicada al LISP estudiando otras funciones de las que suele disponer el lenguaje

Para disponer de una notación uniforme que pueda tratar fácilmente con identificadores, caracteres, números y listas de la misma manera, el LISP se refiere a todo mediante *punteros*. Éstos se pueden simular fácilmente en cualquier lenguaje, incluyendo al BASIC, por ejemplo, si usted imparte la instrucción:

```
POKE 100,55 : POKE 99,100 (la mayoria de los micros)
```

o:

```
?100 = 55 : ?99 = 100 (BBC Micro/Electron)
```

las posiciones 100 y 99 contendrán los valores dados. Se puede considerar que el valor de la posición 99 es un puntero a la posición 100. De modo, pues, que si diéramos la instrucción:

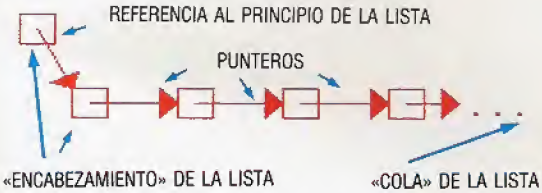
```
PRINT PEEK (PEEK(99))
```

o:

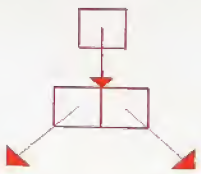
```
PRINT ?(?99)
```

obtendríamos 55 como respuesta. Observe que en algunos micros las posiciones 99 y 100 podrían estar en ROM, de modo que no cambiarían.

Por supuesto, es posible crear punteros hacia cualquier lugar de la memoria, pero si la posición se halla por encima de 255 será necesario combinar dos bytes para obtener la escala entre 0 y 65535. Ahora, si suponemos que, a su vez, el valor 55 es un puntero que señala a otra posición, y así sucesivamente, obtendríamos la estructura de lista indicada.



Lamentablemente, ésta aún sirve de muy poco, puesto que todavía no retiene ninguna información. Ello se consigue combinando posiciones para otorgarle dos punteros a cada nudo.

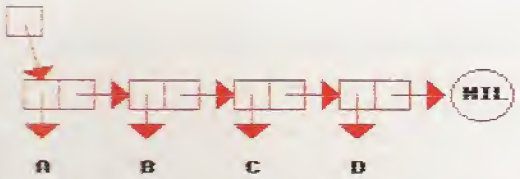


Ahora utilicemos esto para representar la lista:

```
(A B C D E...)
```

Esto está muy bien para listas infinitas, pero debemos ser capaces de detenerlas. Ello se consigue fácilmente haciendo que el puntero final aluda a la lista vacía (o construcción NIL), que ya hemos visto antes. Por tanto, obtenemos la lista finita:

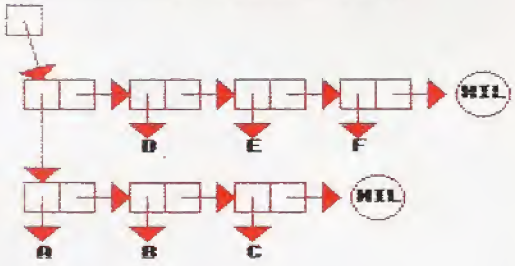
```
(A B C D)
```



Una lista que contiene una lista significa, simplemente, que uno de los punteros de la izquierda señala hacia una nueva lista, en lugar de a un átomo, como en el caso anterior. De modo que la lista:

```
((A B C)D E F)
```

se representaría de la siguiente manera:



A estas alturas ya habrá de estar clara la forma en que trabajan las funciones CAR y CDR. CAR (el encabezamiento de su argumento) no es más que el puntero de la izquierda del primer nudo de su argumento. Por el contrario, CDR (todos a excepción del encabezamiento de su argumento) es simplemente el puntero de la derecha del primer nudo. Así, si llamáramos a la lista completa L, obtendríamos:

```
CONS((CAR L)(CDR L))
```

que da como resultado la lista original L.

Podemos, entonces, ver que CONS es un método para construir listas. Convierte a sus dos argumentos en una nueva lista y devuelve como resultado un puntero al primer elemento de la nueva lista. Mediante CONS se puede crear cualquier estructura

de lista, si bien la cantidad de llamadas CONS puede resultar tediosa, de allí la función abreviada LIST.

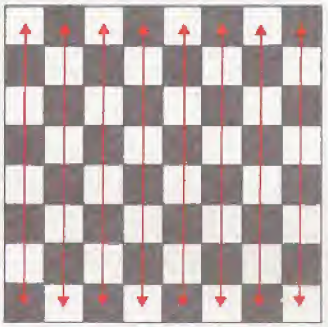
Hemos de destacar que siempre es preferible añadir nueva información a la cabeza de la lista utilizando CONS. Al añadirla a la cola de una lista, es necesario crear una copia de la lista original, pero con el puntero final original señalando hacia el nuevo elemento.

Algunas versiones de LISP poseen una instrucción para esto de forma automática (APPEND), pero ésta no es tan eficaz como CONS. Si la velocidad es esencial, entonces algunas veces es posible utilizar una instrucción para cambiar el puntero requerido en la lista original, en lugar de generar una copia; pero con frecuencia los puristas condenan el empleo de esta clase de «disparates».

Para concluir esta serie, veamos un ejemplo completamente trabajado de un problema muy conocido. El problema consiste en colocar ocho reinas en un tablero de ajedrez normal, de modo tal que ninguna de las reinas se amenace entre sí. Obviamente, ha de haber exactamente una reina en cada columna y una en cada fila. El problema consiste en ordenar tanto las filas como las columnas de modo que ninguna de las reinas se intersecte a lo largo de alguna diagonal.

Problema de las 8 reinas

Imagine que cada columna del tablero de ajedrez contiene una reina. Éstas se ordenarán (del 1 al 8) según sus lugares en una lista llamada REINAS. La fila de la reina (especificada por el valor de REINAS) será variable, y lo calculará el LISP.



REINAS (D1 D2 D3 D4 D5 D6 D7 D8)

El programa comienza con la lista REINAS vacía. En cada etapa se colocará una nueva reina en la fila 1 al comienzo de la lista. La posición de esta reina, y posiblemente de otras, se corregirá (función ALTER) hasta que todas las reinas se encuentren en posiciones seguras. Esta nueva lista se convertirá en REINAS, y se añadirán nuevas reinas al comienzo, y así sucesivamente. Esto continúa hasta que la lista contenga ocho reinas, en cuyo momento se las imprimirá mediante IMP__TABLERO. La función inicial, a la que llamaremos RESOLVER, se puede escribir así:

```
(DEFUN RESOLVER ()
  (SETQ REINAS ())
  (LOOP(UNTIL EQ (LONGITUD (REINAS)8))
    (SETQ REINAS (ALTER (CONS 1 REINAS))))
  (IMP__TABLERO' (DT DC DA D R RA RC
  RT)REINAS))
```

Aquí se puede ver que RESOLVER, sin ningún argumento, establece originalmente la lista vacía REINAS. Luego la función LOOP establece REINAS en el nuevo valor de una lista ALTERada de una reina en la fila 1, seguida por la lista anterior. Este bucle (LOOP) se ejecuta hasta (UNTIL) que la LONGITUD de REINAS sea equivalente a 8. Entonces se imprimirá la lista final.

Aquí hemos utilizado tres nuevas funciones. LONGITUD simplemente devuelve el número de elementos de su argumento, y se define así:

```
(DEFUN LONGITUD (L (N))
  (SETQ N 0)
  (LOOP (WHILE L N)
    (SETQ N (ADD1 N))
    (SETQ L (CDR L)) ))
```

El argumento N, dado entre paréntesis, es la forma que tiene el LISP Acornsoft de mostrar argumentos opcionales. En este caso, N no se le pasa nunca a la función y, por tanto, actúa simplemente como una variable local. La otra función, IMP__TABLERO, combina los elementos de sus dos listas de argumentos para producir una salida comprensible. La función es:

```
(DEFUN IMP__TABLERO (FILAS COLS)
  (COND ((NULL FILAS)T)
    (T (PRINTC (CAR FILAS)(CAR COLS))
      (IMP__TABLERO (CDR FILAS)(CDR  COLS)) )))
```

Se da por sentado que ambas listas son del mismo tamaño, de modo que la función termina si la primera condición es verdadera (una lista vacía). De lo contrario, se imprime el primer elemento de cada lista (con un retorno del carro) y la función se llamará a sí misma para el resto de cada lista.

La tercera función, ALTER, es la que hace todo el trabajo. Se define del siguiente modo:

```
(DEFUN ALTER (REINAS)
  (COND ((EQ (CAR REINAS)9)
  (ALTER (CONS (ADD1  (CADR REINAS))
                      (CDDR REINAS) )))
  ((SEGURAS REINAS)   REINAS)
  (T (ALTER (CONS (ADD1 (CAR REINAS))
                      (CDR REINAS) )))))
```

ALTER funciona comprobando en primer lugar si la reina colocada más recientemente se ha salido por arriba del tablero (fila 9). De ser así, se habrán de mover una o más de las reinas colocadas previamente.

Recuerde que esta reina comenzaba en la fila 1 y que se habrá comprobado en todas las otras filas (mediante la tercera condición de ALTER), de modo que será imposible colocarla con el tablero en la situación actual. Por consiguiente, ALTER se llama a sí misma con el resto de la lista REINAS (todo menos la reina actual).

Al hacer esto, también incrementa la fila de la última reina. Esto se denomina *retroceso* (*backtracking*), y es bastante común en los algoritmos para ordenador de este tipo. Si la segunda condición resulta TRUE (verdadera), entonces todas las reinas situadas en el tablero actual estarán SEGURAS y se devolverá la lista. De lo contrario, se incrementará la posición de la reina colocada en última instancia y se volverá a hacer otro intento.

La única función aún sin definir es SEGURAS, que ha de comprobar que todas las reinas del tablero actual estén seguras. Dado que la posición en la lista REINAS define la columna y que no hay dos

reinas que tengan la misma posición en la lista, no existe necesidad alguna de comprobar columnas que se intersecten. Esto nos deja con la tarea de comprobar sólo las filas y las diagonales. Si suponemos que tenemos una función NOJAQUE que devuelve TRUE (verdadero) si ninguna reina tiene en «jaque» a ninguna de las reinas precedentes, entonces SEGURAS se puede definir así:

```
(DEFUN SEGURAS (REINAS)
 (COND ((NULL REINAS)T)
  (T (AND (NOJAQUE (CAR REINAS)
                   (CDR REINAS) 1)
   (SEGURAS (CDR   REINAS)) ))))
```

Tal vez resulte más fácil comprenderlo definiendo la palabra SEGURAS en términos de sí misma: «La primera reina de la lista REINAS NO tiene en JAQUE a ninguna de las reinas del resto de la lista, AND (y) todas las reinas del resto de la lista están SEGURAS.»

Ahora sólo nos resta definir la función NOJAQUE, que habrá de devolver TRUE si a lo largo de las filas y de las diagonales no se produce ninguna intersección. Ésta será así:

```
(DEFUN NOJAQUE (NUEVA RESTO COL)
 (COND ((NULL RESTO) T)
  (T (AND (NOT (EQ NUEVA (CAR RESTO)))
   (NOT (EQ COL
    (ABS (DIFFERENCE NUEVA (CAR RESTO)))))
   (NOJAQUE NUEVA (CDR RESTO)(ADD1 COL)) ))))
```

Aquí, NUEVA es la reina colocada en último lugar, y RESTO es la lista de las otras reinas del tablero. Suponiendo que en éste haya otras reinas (NULL RESTO es FALSE), la función AND devuelve verdadero si:

1. La fila de la reina actual no es la misma que la fila de la primera reina del resto de la lista.
2. Las reinas no se hallan en la misma diagonal. Esto se comprueba asegurando que la diferencia de fila ABSoluta no sea la misma que la diferencia de COLumna, donde COL retiene la cantidad de columnas que separan a la reina actual de la reina siguiente de la lista (inicialmente 1). En el capítulo anterior ya habíamos definido a ABS como:

```
(DEFUN ABS (NUMERO)
 (COND ((MINUSP NUMERO)(MINUS NUMERO))
   (T NUMERO) ))
```

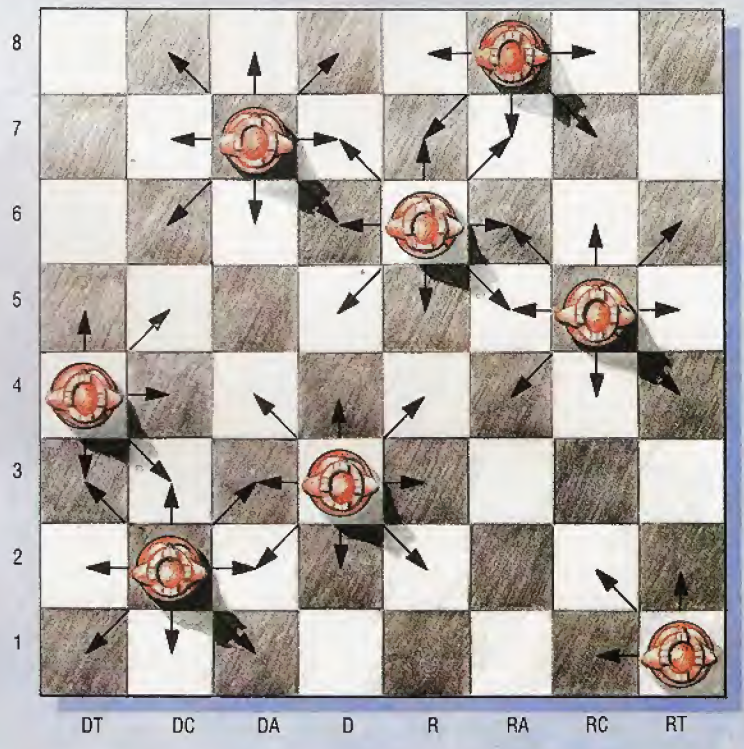
3. 1 y 2 son verdaderos para la reina actual y la siguiente reina de la lista. Esto lo comprueba la función llamándose a sí misma con todos los elementos de RESTO menos el primero e incrementando la diferencia de COLumna.

Ahora ya hemos completado las definiciones necesarias y en la ilustración vemos cómo se ejecutan.

Si usted posee una implementación de LISP, quizá le agrade probar estas rutinas. Recuerde que se han escrito utilizando LISP Acornsoft, si bien no será necesario modificarlas demasiado para que se ejecuten en otros sistemas. El retroceso se puede volver complicado, pero usted puede seguirlo fácilmente imprimiendo la lista REINAS dentro de la función RESOLVER, justo antes de cada llamada a ALTER.

El enfoque de programación de arriba abajo para el «problema de las 8 reinas» ilustra lo fácil que es definir programas en LISP sobre una base de función por función. Por ejemplo, hemos utilizado SEGURAS en ALTER antes de escribirla. Este ejemplo final también debe servir para demostrar la gran versatilidad que posee el LISP.

Un problema real
El problema de las 8 reinas es un ejemplo perfecto de un puzzle que se puede resolver rápidamente merced a las facilidades del LISP para manipulación de listas y funciones recursivas. Antes de examinar atentamente la solución que le ofrecemos en estas páginas, tal vez quiera colocar 8 reinas en un tablero de ajedrez de modo tal que cada reina esté a salvo de ser capturada por alguna de las otras. No se requiere conocimiento alguno de ajedrez, a excepción de que una reina (D) puede moverse en línea recta (diagonal, vertical u horizontalmente) a través de cualquier número de cuadrados, capturando cualquier pieza que encuentre en su camino



Ojo al microdrive

Las rutinas que controlan el funcionamiento de los microdrives del Spectrum son de gran utilidad para el programador en lenguaje máquina

Los accesos al microdrive se llevan a cabo bien por medio de los códigos de enganche ya vistos, o bien llamando directamente a las rutinas ROM de la Interface 1. Pero esto último no es muy recomendable, toda vez que pueden existir varias ROM de la Interface 1. Examinaremos, pues, los trece códigos de enganche dispuestos para que podamos usar el microdrive.

Antes de hacer este examen vamos a revisar algunos principios del manejo de un microdrive. Una vez formateado un cartucho, queda dividido en 255 *sectores,* que pueden ser considerados como las unidades físicas de almacenamiento de información. Cada sector se compone de un encabezamiento (datos que, entre otras cosas, incluyen el nombre del cartucho), un bloque de datos, que contiene el nombre del fichero que está usando ese sector, algunos otros datos y un registro de 512 bytes. El formateo pone marcas a todos los sectores de la cinta que no se pueden utilizar.

Al igual que todas las demás comunicaciones con los dispositivos de E/S del sistema del Spectrum, la comunicación microdrive-Spectrum se lleva a cabo por medio de un canal. Para los microdrives, éste toma la forma de un área de memoria de 295 bytes. Asociado a un canal de microdrive existe un mapa de 32 bytes para microdrives, establecido por medio del sistema operativo para indicar a las diferentes rutinas cuál de los sectores queda libre para su uso. Éste contiene una indicación sobre los sectores no utilizables y los que ya han sido utilizados para el almacenamiento de más información.

El canal se almacena en el área de memoria llamada *microdrive channel area* (área del canal de microdrive). Siempre que se establece un canal, la memoria que está entre el final del área del canal de microdrive y STKEND se desplaza hacia arriba dentro de la memoria para dejarle espacio. Igualmente, cuando ya no se necesita el canal, se cierra y baja de nuevo dicha memoria.

Los canales del microdrive son establecidos por el sistema de dos maneras: la *explícita,* cuando se requiere un fichero por la instrucción OPEN, y la *implícita*, cuando el sistema tiene que abrir un canal para realizar una operación. Es esto lo que hace la instrucción SAVE del microdrive: la apertura del canal se dice «implícita» dado que se sobreentiende que el canal debe estar abierto para ejecutar la instrucción, aun cuando no lo empleemos directamente. Un canal abierto explícitamente ha de ser cerrado manualmente, mientras que el cierre es automático en un canal abierto de forma implícita.

Ofrecemos en este capítulo un gráfico que muestra la estructura del canal del microdrive. Notará sin duda la semejanza entre los primeros bytes de este canal y los canales que establecemos para enviar información a la pantalla o leer datos desde el teclado. Buena parte de esta información no es estrictamente necesaria para el programador, ya que generalmente no hay que preocuparse de cómo se accede directamente a la información del canal del microdrive.

La dirección del mapa del microdrive está contenida en el canal, y podemos averiguar el espacio disponible en un cartucho por medio del examen del mapa de este cartucho. De hecho es así cómo la función del BASIC CAT proporciona información sobre la cantidad de espacio que queda en el cartucho. Si desea explorar el mapa del microdrive, pruebe la siguiente rutina (pulse previamente NEW).

```
 10 OPEN #5;"M";1;"testprog"
 20 start=PEEK(23870)+256*PEEK(23871)
 30 FOR I=start TO start+31
 40 LET sector=PEEK(I)
 50 FOR J=1 TO 8
 60 IF sector/2=INT(sector/2) THEN PRINT "0";
 70 IF sector/2<>INT(sector/2) THEN PRINT "1";
 80 LET sector=sector/2:LET sector=INT(sector)
 90 NEXT J
100 PRINT
110 NEXT I
120 CLOSE #5
```

Al ejecutar este programa, el 1 significa sector usado o no utilizable y el 0 sector libre. Puede que le interese alterar este programa para visualizar la cantidad de espacio libre en el cartucho.

Respecto a la sección enorme, de 512 bytes, para datos dentro del dibujo sobre la estructura de los canales, el OS lo llena con los datos a grabar. Los datos se escriben en el cartucho bien cuando el buffer está lleno, bien cuando se ejecuta una instrucción CLOSE# (o su código de enganche equivalente). En el último caso, el buffer es escrito en el cartucho quedando ese registro marcado como un EOF (*end of file:* fin de archivo). Una tercera posibilidad de escritura la da el empleo de un código de enganche, como veremos. Bajo condiciones normales, se escribirá en el cartucho a intervalos de 512 bytes.

Analicemos ahora los códigos de enganche empleados para controlar el microdrive.

Códigos de enganche del microdrive

- *Código de enganche 33:* enciende el motor en una unidad determinada del microdrive («selección del drive», suele decirse). Los microdrives están numerados de uno a ocho. La rutina puede servir

también para desconectar los motores de todos los microdrives. Y aquí es importante notar que la rutina puede ejecutar un Return con las interrupciones desactivadas. Así que, si usted emplea esta rutina, a su vuelta lo primero que habrá de hacer es generar la instrucción EI.

El empleo de la rutina es sencillo. Primero hay que disponer el registro A para que retenga el número del drive que desea poner en marcha. Después empleará RST#8 para llamar al código de enganche. Poniendo el valor 0 en el registro A se desconectarán los drives. De otra forma también pueden desconectarse: generando un error en BASIC. Los errores pueden generarse por medio de este código de enganche de dos maneras: tratando de poner en marcha un drive no conectado, o bien si se enciende un drive que no contenga un cartucho formateado.

• *Código de enganche 34:* Esta utilísima llamada nos permite abrir un fichero en un cartucho. Pronto estudiaremos cómo establecerla. Pero una vez llamada, la rutina examinará en el drive si existe un fichero con el nombre que usted le ha dado. Si encuentra uno, el fichero será abierto para *lectura*. Si no lo encuentra, será abierto para *escritura*. Esto se indica con el estado del flag de Escritura/Lectura del canal que establece esta llamada: el bit 0 valdrá 0 para leer un fichero y 1 para escribirlo.

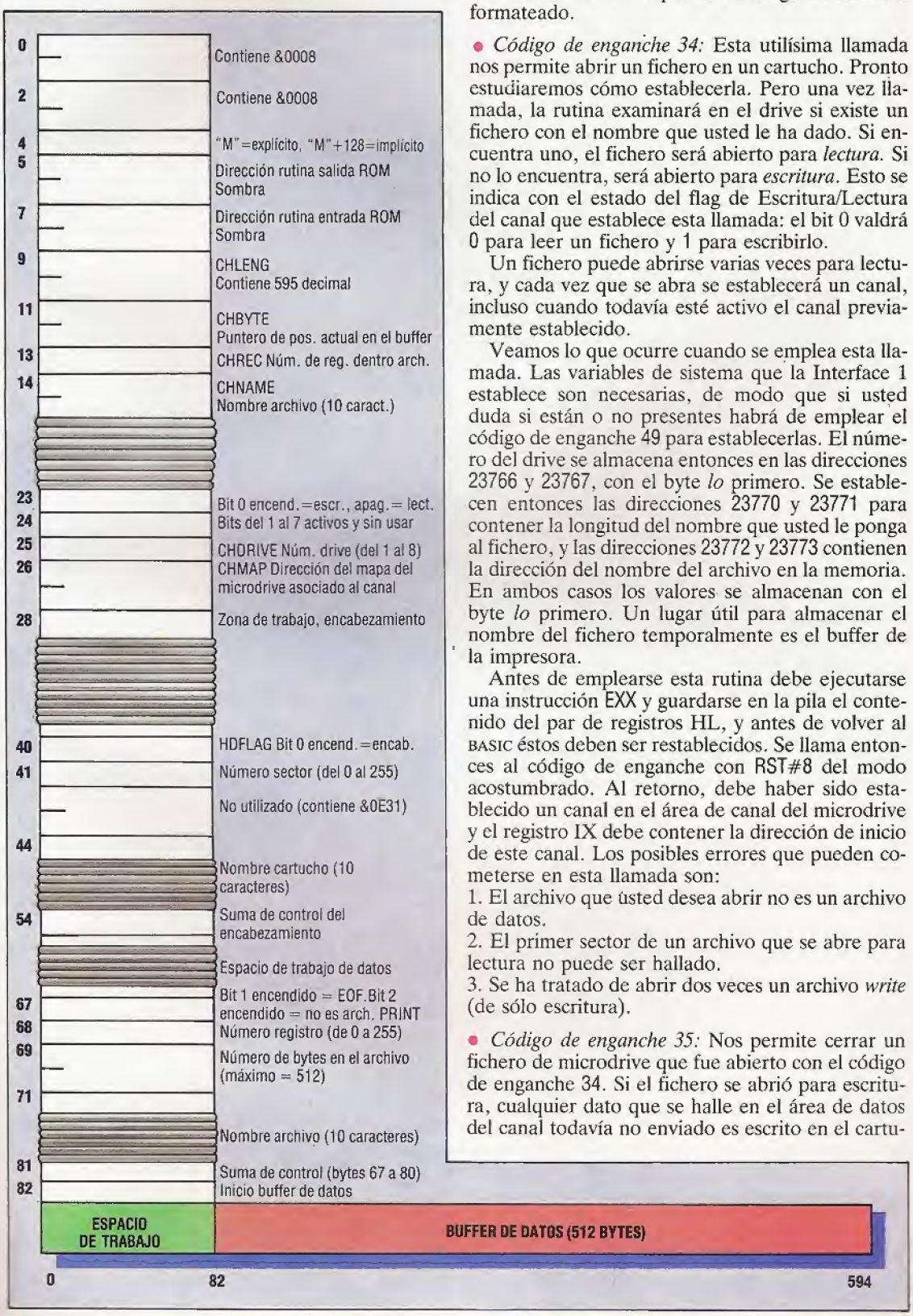
Un fichero puede abrirse varias veces para lectura, y cada vez que se abra se establecerá un canal, incluso cuando todavía esté activo el canal previamente establecido.

Veamos lo que ocurre cuando se emplea esta llamada. Las variables de sistema que la Interface 1 establece son necesarias, de modo que si usted duda si están o no presentes habrá de emplear el código de enganche 49 para establecerlas. El número del drive se almacena entonces en las direcciones 23766 y 23767, con el byte *lo* primero. Se establecen entonces las direcciones 23770 y 23771 para contener la longitud del nombre que usted le ponga al fichero, y las direcciones 23772 y 23773 contienen la dirección del nombre del archivo en la memoria. En ambos casos los valores se almacenan con el byte *lo* primero. Un lugar útil para almacenar el nombre del fichero temporalmente es el buffer de la impresora.

Antes de emplearse esta rutina debe ejecutarse una instrucción EXX y guardarse en la pila el contenido del par de registros HL, y antes de volver al BASIC éstos deben ser restablecidos. Se llama entonces al código de enganche con RST#8 del modo acostumbrado. Al retorno, debe haber sido establecido un canal en el área de canal del microdrive y el registro IX debe contener la dirección de inicio de este canal. Los posibles errores que pueden cometerse en esta llamada son:

1. El archivo que usted desea abrir no es un archivo de datos.
2. El primer sector de un archivo que se abre para lectura no puede ser hallado.
3. Se ha tratado de abrir dos veces un archivo *write* (de sólo escritura).

• *Código de enganche 35:* Nos permite cerrar un fichero de microdrive que fue abierto con el código de enganche 34. Si el fichero se abrió para escritura, cualquier dato que se halle en el área de datos del canal todavía no enviado es escrito en el cartu-



Estructura del canal del microdrive
El canal del microdrive ocupa 595 bytes de memoria. Su posición debe estar contenida en el registro IX después de abrir un fichero en el cartucho por medio del cód. de enganche 34

Kevin Jones

Dar vida

Robyn Beeche

Entre los postulados de la robótica y la realidad actual existe aún un profundo abismo... Analicemos el tema

En ningún otro campo de la informática existe un vacío tan grande entre la realidad y la fantasía como el que existe en el campo de la robótica. Los robots que vemos en las películas de ciencia-ficción rodadas en Hollywood caminan, hablan y traman la destrucción de la humanidad. Sin embargo, a escasos kilómetros de «la capital del cine», en la Stanford University, los últimos robots de investigación apenas si pueden deambular a través de una habitación sin darse de bruces contra los muebles.

Por supuesto, los robots desempeñan un papel productivo en las fábricas de automóviles y en la industria en general. Pero tales sistemas, generalmente para aplicar pintura o soldadura por arco, se limitan a repetir secuencias de operaciones preprogramadas sin ninguna variación. Son tan «tontos»

«Starlet» cibernética

Los robots prácticos se consideran un desarrollo moderno, pero Rosa Bosom (*Radio-Operated Simulated Actress Battery Or Stand-by Operated Mains*), a quien vemos aquí fotografiada junto a su inventor, Bruce Lacey, fue construida en 1966 para hacer el papel de reina de Francia en una producción de *Los tres mosqueteros* en el Royal Court Theatre de Londres. Construida con relés y motores excedentes del gobierno, Rosa acepta instrucciones en forma de tonos musicales, que se convierten en movimiento, y también está equipada con sensores ultrasónicos para detectar obstáculos y evitarlos. En su carrera de actriz, Rosa remedaba sus palabras con voz grabada en cinta, mientras sus labios se movían por control remoto. Lo más interesante, desde el punto de vista cibernético, es que Rosa puede interactuar con un segundo robot llamado Mate. Rosa ha aparecido en varias exhibiciones, incluyendo la Cybernetics Serendipity en el ICA, y más recientemente ganó en Londres el concurso Alternative Miss World (Miss Mundo alternativa)

que, de producirse un vacío en la cinta transportadora que les alcanza el trabajo, arrojarían pintura a ciegas o soldarían en el aire.

Las razones de esta abismal disparidad entre PR y rendimiento son complejas, pero se pueden resumir en dos palabras: *percepción* y *planificación*. En primer lugar, los robots de hoy en día no sacan ningún «sentido» de sus sensores. En segundo lugar, aunque lo hicieran, no sabrían qué hacer. Ésta es la razón por la cual la robótica es un campo de pruebas tan idóneo para aplicar las teorías de inteligencia artificial (AI).

La robótica toca todas las facetas de la AI porque su objetivo es producir un artefacto que pueda enfrentarse con el mundo real. Por consiguiente, un robot ha de extraer algún significado de su entorno.

No es demasiado difícil dotar a un ordenador de toda una variedad de dispositivos de entrada (cámaras de televisión, sensores de calor, exploradores ultrasónicos, teclados de presión, etc.) que puedan darle acceso a una información que los seres humanos no podemos obtener directamente, como la luz infrarroja o la ultravioleta. Pero a menos que el robot habite en un entorno severamente controlado, no comprenderá el significado de sus sensores. La percepción humana, como hemos visto en el capítulo sobre la visión por ordenador, es un proceso permanente de modelación psicológica que se basa en lo que está sucediendo en el mundo real.

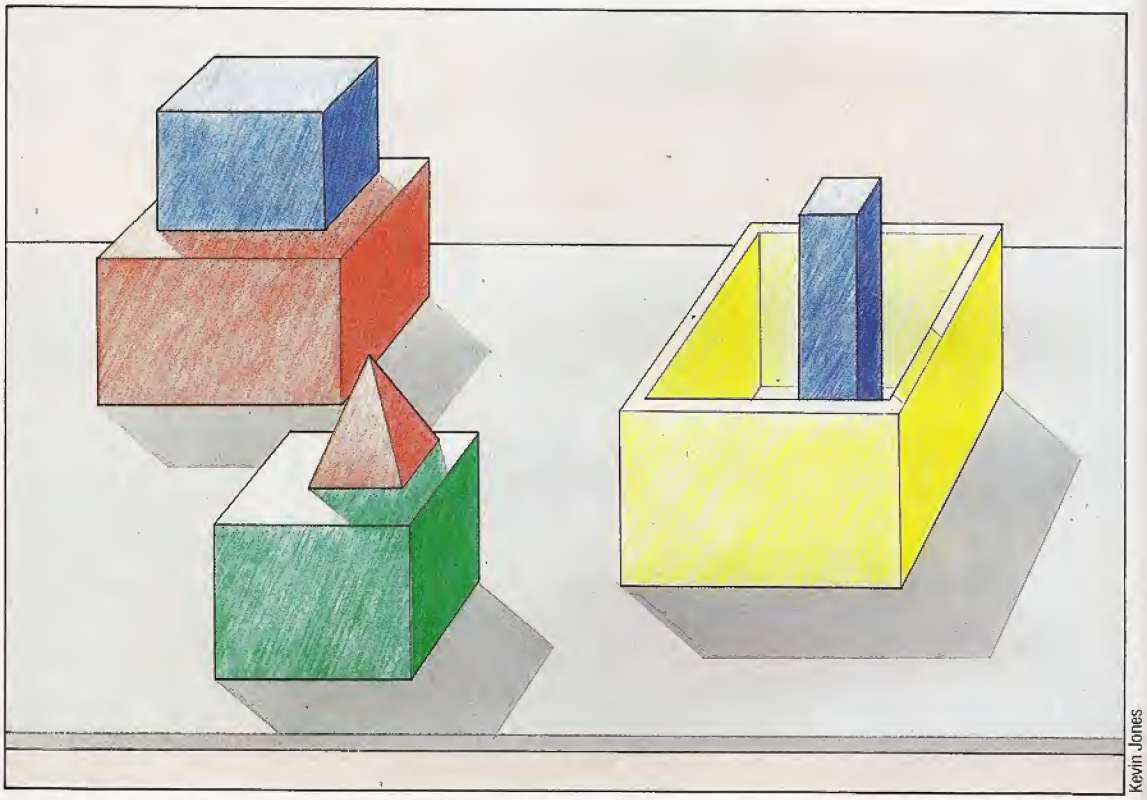
Una cosa es idear un algoritmo eficaz para hallar una ruta a través de un laberinto retenido en la memoria de un ordenador y visualizado en una pantalla, y otra cosa muy distinta utilizar ese algoritmo para conducir un robot a través de un entorno urbano, trozos de setos demasiado crecidos u otros obstáculos insospechados a lo largo del camino.

En el mundo real, la ley suprema es la de Murphy. A ello se debe la enorme popularidad de las competiciones de Micromouse. En estas competiciones, un «ratón» controlado por ordenador ha de navegar hasta el centro de un complejo laberinto de mesa. Desde hace ya muchos años se conocen métodos para hacerlo, al menos en teoría. En la práctica, sin embargo, las paredes nunca son totalmente rectas, quedan manchas resbaladizas allí donde derramaron aceite los contendientes anteriores, etc. En tales circunstancias, la robustez y la adaptabilidad importan más que la elegancia algorítmica.

En la mayoría de los otros campos de la AI, los programadores pueden refugiarse en «microcosmos» creados por ellos mismos, como sucede en el caso de un programa para jugar al ajedrez, que opera en un entorno abstracto. Un sistema experto se ocupa de hechos, no de cosas. Esto explica por qué a quienes investigan en el campo de la AI les atrae tanto el llamado *mundo de bloques*.

El *mundo de bloques* es un entorno esquemático simplificado que contiene bloques de construcción de colores de carácter infantil. No son auténticos bloques de construcción, por supuesto, sino *representaciones* formalizadas de esos bloques. Es el lugar predilecto de los científicos dedicados al desarrollo de la AI, cuyos programas pueden resolver problemas y manipular objetos dentro de ese mundo sin renunciar a la comodidad de sus CPU.

Los investigadores en robótica no cuentan con ningún refugio de este tipo. Los robots deben enfrentarse a las vicisitudes de la realidad lo mejor que puedan. Un robot inteligente (uno que se pueda desplazar por su propia iniciativa) debe ser capaz de construir un modelo cognoscitivo de su entorno. Además (para satisfacer décadas de literatura y cine de anticipación) ha de responder a instrucciones habladas. Y puesto que sus respuestas



Kevin Jones

No es el mundo real
El *mundo de bloques* es un modelo sumamente abstracto del mundo real, que permite a los científicos dedicados a la investigación en el campo de la inteligencia artificial evaluar diferentes métodos de análisis de escena, comprensión de textos y solución de problemas. Sin embargo, puede ser difícil traducir con éxito los métodos del mundo de bloques al mundo real

no se pueden prever, sería muy deseable que tuviera cierto grado de capacidad de aprendizaje. Éste es el motivo por el cual realmente no son robots *inteligentes*. Antes de que se pueda construir uno, se deben resolver estos problemas de AI:

- Visión por ordenador.
- Comprensión del habla.
- Resolución de problemas en un entorno dinámico.
- Aprendizaje de la máquina.

y ésta es la lista mínima.

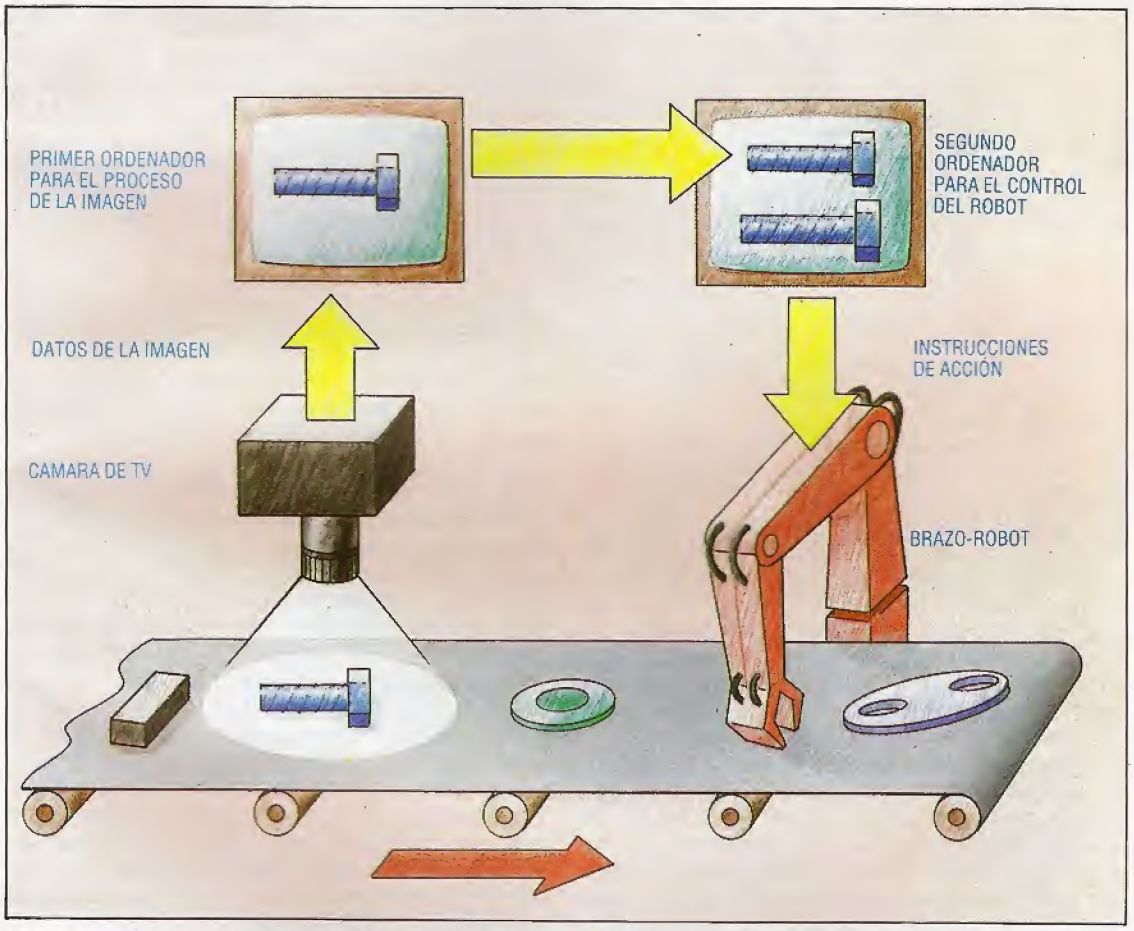
Los robots industriales modernos tienden a diseñarse para entornos sumamente estructurados. Por lo general, un robot no es más que un dispositivo similar a un brazo bajo el control de un ordenador, integrado en una factoría automatizada o en un banco de taller. Su «mano» puede tener atornilladas herramientas y un operador humano puede «enseñarle» una serie de movimientos que el robot puede repetir cuantas veces se precise. Puede memorizar tales secuencias de acciones, pero no modificarlas. Con frecuencia opera en un ambiente regulado tan estrechamente que no necesita en absoluto capacidades perceptuales.

Algunos robots, sin embargo, han dado un pequeño paso hacia el mundo real. Son un poco menos dependientes de las personas u otras máquinas que les proporcionen su trabajo. Pueden, hasta cierto punto, ir a buscarlo ellos mismos.

La forma más común de dotar a un robot de un sentido es a través de la vista. Una cámara de video toma imágenes (de un objeto que el robot haya de recoger, p. ej.) y pasa la información de imagen al ordenador de control (éste puede ser un ordenador separado del que controla el brazo del robot). El ordenador procesa la información visual proporcionada (mediante el empleo de métodos de arriba abajo o de abajo arriba, o tal vez de una combinación de ambos) y detecta ciertos patrones que son importantes en el contexto de la tarea del autómata informatizado. Esta información se envía al ordenador que controla el brazo, donde actualiza una descripción simbólica de su entorno ya almacenada allí. El robot posee ahora una imagen de los objetos y las herramientas con las que está trabajando, e instrucciones sobre lo que ha de hacer.

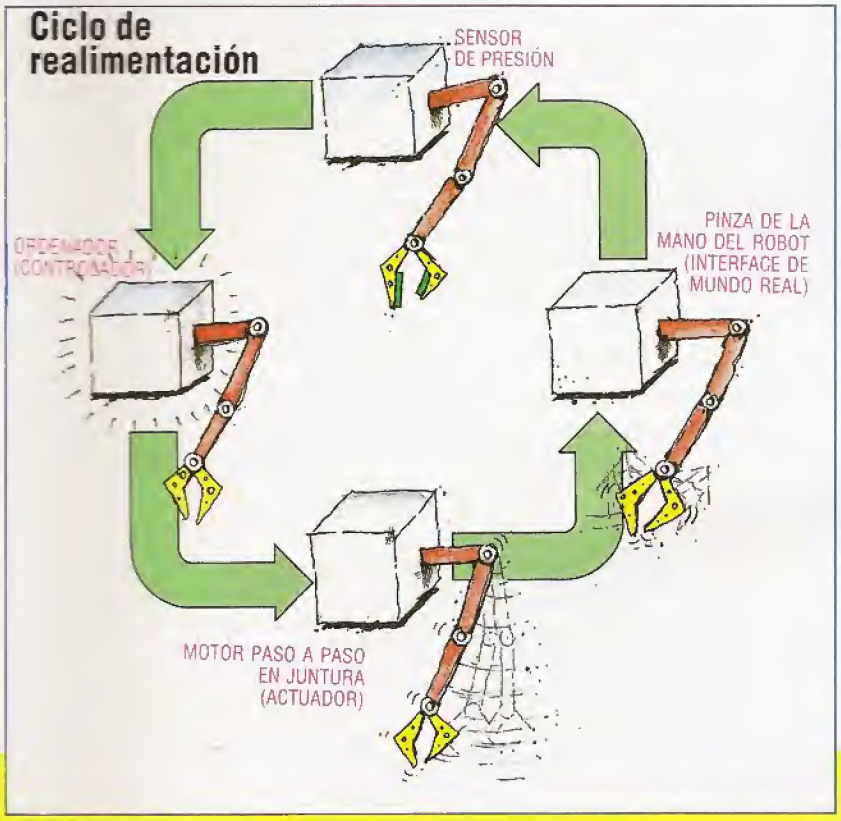
La gran ventaja de un robot dotado de visión, por limitada que ésta sea, es que al autómata informatizado se le pueden proporcionar componentes en diversas orientaciones, y posiblemente también componentes de formas diferentes. Está en condiciones de reconocer el tipo de componente del que se esté ocupando y el ángulo en el cual ha llegado, de modo que podrá manipularlo de modo correcto. Un robot «ciego» por completo intentaría coger cada elemento de la misma manera, probablemente con resultados desastrosos.

Otra forma de hacer que los robots sean más «sensibles» es a través del tacto. Los ingenieros pueden instalar sensores de fuerza o de presión en el extremo de la mano del robot. Cuando éstos entran en contacto con algún objeto, la información se retransmite al ordenador controlador. Si el robot se topa con una pared, el sensor de fuerza volverá a enviarle información de modo que pueda modificar la trayectoria de su brazo para evitar que se repita el accidente.



Kevin Jones

Transportando la idea
Un robot con capacidad de visión puede utilizarse para reconocer y recoger ciertos elementos de una cinta transportadora. En este caso el autómata informatizado necesitará dos ordenadores: uno para interpretar los datos provenientes de la cámara de televisión, y otro para controlar el mecanismo de prensión



Mike Clowes

Dedos ágiles
Para manipular objetos con precisión se necesita el sentido del tacto. Siempre que recogemos objetos, estimamos el peso del objeto con el fin de determinar la fuerza de agarre que se requiere para poder levantarlo con éxito. El proceso, que se ha reproducido con niveles de éxito variables en las aplicaciones de robots, esencialmente es un ciclo de realimentación en el cual sensores táctiles vuelven a alimentar los datos al ordenador, que a su vez aprieta o afloja el agarre sobre el objeto

La exhibición que se realiza comúnmente para promocionar los robots sensibles al tacto consiste en hacer que primero perforen un agujero en un trozo de acero fresado y después hacerle realizar lo mismo pero con un huevo, sin romper su cáscara.

Tanto la visión como el tacto, aunque en estadios de desarrollo primitivos en los robots actuales, demuestran la importancia de la realimentación. Para que los robots sean útiles en situaciones diversas es esencial proporcionarles capacidades de realimentación. Por supuesto, es muy difícil transformar información sobre el mundo exterior a una forma que el ordenador del robot pueda utilizar para guiar sus acciones. Uno de los muchos problemas es el de transformar los datos a «tiempo real» (con suficiente rapidez para que el robot responda a lo que está sucediendo antes de que el evento percibido haya terminado). Se dice que el escaso número de robots que están equipados con dispositivos sensoriales y alguna capacidad para interpretar la realimentación son robots de la *segunda generación*.

Ya hemos visto que la comprensión del habla continua era el más arduo de los cuatro tipos de problemas del proceso del lenguaje. La principal dificultad estriba en que el ordenador no posee control alguno sobre lo que se le está diciendo.

No obstante, se puede otorgar al robot un grado útil de reconocimiento de voz (en contraposición a la comprensión del habla). Esto significa tan sólo que se puede entrenar al robot para que responda de la forma adecuada a una pequeña cantidad de palabras y frases. Su comprensión de lo que se le dice es menor que la de un perro que sepa sentarse, tumbarse o ir a buscar algún objeto, pero en una factoría en la cual los obreros humanos pueden tener sus manos ocupadas, incluso estas capacidades lingüísticas tan simples pueden ser de utilidad.

Algunos de los desarrollos más interesantes que se han producido en el campo de la robótica han sido consecuencia de la exploración espacial. Ingenieros de la NASA han diseñado un Mars Rover, un vehículo robótico que puede rodar sobre la superficie del planeta Marte. El vehículo dispone de sensores, como cámaras de televisión, para recoger información sobre sus inmediaciones, y la inteligencia suficiente para hacer uso de la misma. No le es dable esperar señales provenientes de la Tierra antes de decidir lo que tiene que hacer a continuación, pues los mensajes enviados por radio pueden tardar hasta media hora en hacer el recorrido completo. El Mars Rover está planeado para el futuro, pero en 1976 los norteamericanos hicieron aterrizar en Marte dos sondas Viking. Éstas pudieron analizar muestras del suelo y de la atmósfera marciana sin la supervisión paso a paso por parte del «control de misión».

Colocar robots en el espacio para que trabajen en lugar de seres humanos supone un considerable ahorro de dinero. Según las últimas estimaciones norteamericanas, mantener a una persona en el espacio y traerla de vuelta sana y salva ¡cuesta alrededor de 10 000 dólares la hora! Quizá resulte oneroso desarrollar robots espaciales, pero a la larga serán más baratos que sus equivalentes humanos.

Una posibilidad exótica para el futuro es la idea que presentó por primera vez John von Neumann a finales de los años cuarenta. Von Neumann, el matemático que sentó los cimientos teóricos para el ordenador digital electrónico, hizo además otros muchos trabajos, incluyendo el desarrollo de una teoría de autómatas capaces de reproducirse a sí mismos. Él propuso que las máquinas podían reproducirse siguiendo un sencillo conjunto de reglas. Según el esquema de Von Neumann, un sistema robótico autorreproductivo necesita cuatro componentes. El primero es una factoría automatizada que reúna la materia prima y la convierta en productos según las instrucciones dadas. El segundo componente es un duplicador que copie estas instrucciones. El tercero es un controlador que pase las instrucciones al duplicador para que las copie. El cuarto y último componente es el conjunto de las propias instrucciones, que le dice al sistema cómo construir una nueva factoría automática completa a partir de los productos que haya fabricado.

Durante casi 40 años éstos fueron apenas conceptos teóricos, hasta que los investigadores de la NASA propusieron un anteproyecto de ingeniería para un sistema que se duplicara a sí mismo en la Luna. Tal factoría espacial contendría un «constructor universal» que tomaría los componentes fabricados por la unidad de producción y construiría con ellos una nueva factoría, incluyendo, por supuesto, otro constructor universal. Se valdría de las materias primas de la Luna y no requeriría servicios de mantenimiento desde la Tierra. Asimismo, podría proveerse a sí mismo de un buen contingente de nuevos obreros robot.

Puesta a punto

Añadiremos algunas rutinas para manipular la pila, además de las útiles implementaciones de la recursión

Cuando se escribe un programa de este tipo, por lo general se necesitan algunas rutinas de utilidad general.

Entre éstas se incluyen procedimientos para actualizar el tablero, comprobar la legalidad de los movimientos, etc. Por ejemplo, en un programa para jugar al ajedrez sería muy útil una rutina para comprobar si el rey está o no en jaque.

Por supuesto, podríamos escribir una rutina para hallar movimientos que pongan en jaque al rey del adversario y, durante la defensa, posiciones que mantengan a nuestro rey a salvo de tal eventualidad. Sin embargo, tiene mucho más sentido contar con una rutina general que asigne un valor TRUE o FALSE a "¿está el rey de COLor en jaque?". Luego podemos llamar a la rutina con diferentes valores de COLor y decidir después qué hacer con el resultado.

Para nuestro programa de *go* vamos a adoptar un enfoque similar. En lugar de tener rutinas específicas que decidan qué hacer con diversos tipos de grupos, implementaremos una rutina general para contar los miembros de cualquier grupo determinado. Entonces podremos decidir, en las rutinas de llamada separadas, qué significado tienen los resultados.

El procedimiento principal para hacer esto se denomina PROCbuscar en la versión para el BBC Micro. Se le pasan dos parámetros: la posición de todas las fichas del grupo que deseemos contar (P%) y el color del grupo de fichas (C%). Luego el procedimiento le suma 1 a CSTN%, que se utiliza para totalizar la cantidad de fichas del grupo, y observa las cuatro fichas adyacentes. Para cada una de estas fichas adyacentes, lleva a cabo el mismo procedimiento, pero sólo si la ficha no se ha contado todavía. Por supuesto, ya disponemos de un procedimiento para llevar a cabo esta búsqueda en cuatro direcciones desde una ficha dada, ¡de modo que la forma obvia de contar las fichas adyacentes consiste en que el procedimiento se llame a sí mismo! Éste es un ejemplo de recursión y puede que a primera vista parezca complicado. Para esclarecer un poco las cosas, he aquí un análisis estructurado de la rutina de búsqueda:

```
BUSCAR (desde-posición,para-color)
 IF desde-posición se halla en el tablero
  AND desde-posición contiene una ficha de para-color
  AND todavía no hemos contado desde-posición
 THEN
  marcar desde-posición, para que no volvamos a
  contarla
  sumar uno a CSTN% (nuestra variable del contador)
  BUSCAR (norte de desde-posición, para-color)
  BUSCAR (este de desde-posición, para-color)
  BUSCAR (sur de desde-posición, para-color)
  BUSCAR (oeste de desde-posición, para-color)
 ENDIF
END BUSCAR
```

Observe que las llamadas recursivas se tratan como cuatro sentencias separadas, y no en un bucle, para evitar problemas cuando la recursión es profunda.

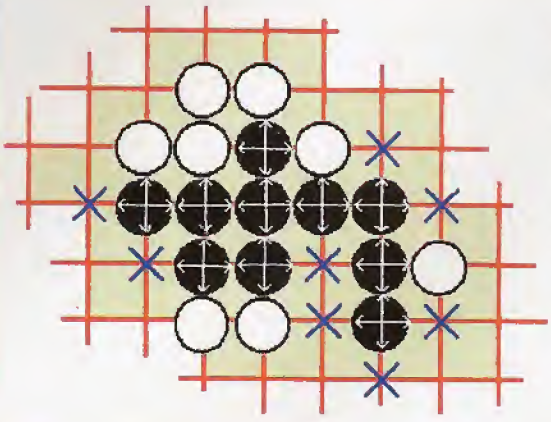
La estructura de pila

Cuando la rutina se llama a sí misma, el parámetro *desde-posición* se reemplaza por la nueva *desde-posición* (en una de las cuatro direcciones) y la rutina prosigue. La nueva *desde-posición* también puede llamar de forma recursiva a la rutina de búsqueda, creando otra variable *desde-posición,* y así sucesivamente. En algún punto, una de las condiciones IF será falsa, de modo que la llamada en cuestión terminará sin que las sentencias THEN realicen una llamada recursiva. Siempre que la recursión retorna de un nivel como éste, se restablece el valor anterior de *desde-posición.*

Se puede pensar en esta estructura de datos de «pila» como en una pila de platos. Cuando la rutina es llamada recursivamente, por arriba de la pila se añade un nuevo plato. Sin embargo, cuando acabe este nivel particular de la recursión, este mismo plato se debe quitar de la pila (restableciéndose el mismo valor de variable). Esta estructura se denomina LIFO (*Last In, First Out*: último en entrar, primero en salir).

Lamentablemente, aunque el BASIC BBC admite la recursión en el pase de parámetros de PROCedimientos y variables localizadas, el Commodore 64, Sinclair Spectrum y Amstrad CPC 464/664 carecen de esta facilidad. Por consiguiente, en estas rutinas usted encontrará las matrices llamadas SK%() o s(), que "apilan" los valores necesarios.

Ello se realiza reteniendo un puntero (PILA% o su equivalente) que llevan un registro de la posición



Contando cabezas

La rutina de búsqueda, que se utiliza para evaluar el estado de los grupos del tablero, incorpora (o simula) procedimientos recursivos para buscar en las cuatro direcciones desde cada una de las fichas del grupo. La rutina termina cuando se llega a una licencia o a una ficha del enemigo

superior de la pila. Cada vez que se coloque algo en la pila, se incrementa este puntero, y se lo reduce cada vez que se saca algo. De esta manera, la variable adecuada se restablece al valor de la parte superior de la pila.

Un efecto secundario de esta rutina de búsqueda es que la recursión siempre termina en los márgenes del grupo que esté examinando. Cuando buscamos estos márgenes, podemos comprobar si se hallan en el margen del tablero o si contienen una ficha del color contrario. Si ninguna de estas condiciones es verdadera, necesariamente ha de ser una licencia del grupo. Por consiguiente, la segunda parte de esta rutina cuenta la cantidad de licencias relacionadas con el grupo, utilizando la variable CLIB%. Nuevamente, estas posiciones se han de marcar tras haberlas contado, ya que de lo contrario se correría el riesgo de contarlas dos veces.

Al contar un grupo de fichas, siempre llamaremos a la rutina PROCcontar, que a su vez llama a PROCbuscar. Ésta inicializará las variables del contador y limpiará los marcadores tras utilizar la rutina de búsqueda.

Al marcar posiciones, observará que empleamos el propio tablero real, para evitar las cuentas dobles. No obstante, tras haber marcado el tablero, hemos de poder suprimir los marcadores antes de continuar. Nuevamente hemos utilizado una rutina general (PROClimpiar), que opera mediante AND los bytes del tablero apropiados con 3, que suprime el marcador del bit 2. Puesto que los dos bits menos significativos de cada byte del tablero contienen los colores, y que los marcadores de ficha y licencia están retenidos en el tercero y el cuarto bit, la llamada PROClimpiar(3) es ideal.

Por ejemplo:

Byte:	A	B	C	D	E	F	G	H
Máscara:	0	0	0	0	0	0	1	1
Resultado:	0	0	0	0	0	0	G	H

La versión para el Spectrum tiene un aspecto diferente, porque en el Spectrum la instrucción AND no desenmascara los bits en los que no estamos interesados. Sólo se puede utilizar para dar resultados verdadero/falso en expresiones tales como: IF X<3 AND Y=3 THEN... En consecuencia, hemos de recurrir a una operación de resta aritmética normal para limpiar el bit 2 en la versión de esta rutina para el Spectrum.

Definiendo esta rutina de limpieza general, también podemos utilizarla para limpiar el tablero antes de una partida, llamando a la rutina con parámetro cero. Esto se ha añadido en la línea 1400.

Sólo nos resta un procedimiento del juego antes de que podamos comenzar a escribir las verdaderas rutinas para procesar los movimientos. Se trata del código que organiza el tablero al comienzo del juego con las adecuadas fichas de handicap. Tal como explicáramos en el primer capítulo de esta serie, el *go* posee un método exclusivo para equilibrar los niveles de destreza, al permitir que el jugador más débil (quien juega siempre con las negras) coloque entre dos y nueve fichas en el tablero como su primer movimiento. Estas fichas se han de colocar en posiciones específicas, que se establecen en la rutina leer handicaps (líneas 600-750). Si usted ejecutó la primera parte del programa que ofrecimos anteriormente, habrá observado que se le indicó que entrara un número de handicap al final de la rutina pantalla_titulos (pero entonces ese número era ignorado). Añadiendo la línea 1580 y la rutina desde la línea 1630 a la 1690, ahora las fichas de handicap se colocarán correctamente; las sentencias DATA entre las líneas 670 y 740 dan las posicio-

Módulo dos

BBC Micro

```
270 PROCleer_handicaps
590 :
600 DEF PROCleer_handicaps
610 LOCAL L%
620 DIM hncp% 43
630 RESTORE 670
640 FOR L% = 0 TO 43
650   READ hncp%?L%
660   NEXT
670 DATA &44,&CC
680 DATA &44,&CC,&4C
690 DATA &44,&CC,&4C,&C4
700 DATA &44,&CC,&4C,&C4,&88
710 DATA &44,&CC,&4C,&C4,&84,&8C
720 DATA &44,&CC,&4C,&C4,&84,&8C,
    &88
730 DATA &44,&CC,&4C,&C4,&84,&8C,
    &48,&C8
740 DATA &44,&CC,&4C,&C4,&84,&8C,
    &48,&C8,&88
750 ENDPROC
760 :
770 REM ****************************************
1400 PROClimpiar(0)
1580 PROChandicap(hand%)
1620 :
1630 DEF PROChandicap(hand%)
1640 LOCAL L%,P%,Q%
1650 Q%=INT((hand%-2)/2*(hand%+1)+0.5)
1660 FOR L%=Q% TO Q%+hand%-1
1670   P%=hncp%?L% : tablero%?P%=negras%
1680   NEXT
1690 ENDPROC
1700 :
1710 REM ****************************************
4030 :
4040 DEF PROCcontar(P%,C%)
4050 clib%=0 : cstn%=0
4070 PROCbuscar(P%,C%)
4080 PROClimpiar(color%)
4090 ENDPROC
4100 :
4110 REM ****************************************
4120 :
4130 DEF PROCbuscar(P%,C%)
4140 IF (P% AND 240)=0 OR (P% AND 15)=0 THEN
       ENDPROC
4150 IF (tablero%?P% AND color%)=0 THEN
       4250
4160 IF (tablero%?P% AND C%)=0 THEN
       ENDPROC
4170 IF (tablero%?P% AND marcador%)>0 THEN
       ENDPROC
4180 tablero%?P%=C%+marcador%
4190 cstn%=cstn%+1
4200 PROCbuscar(P%+dir%(1),C%)
4210 PROCbuscar(P%+dir%(2),C%)
4220 PROCbuscar(P%+dir%(3),C%)
4230 PROCbuscar(P%+dir%(4),C%)
4240 ENDPROC
4250 IF (tablero%?P% AND licencia%)>0 THEN
       ENDPROC
4260 tablero%?P%=licencia%
4270 clib%=clib%+1
4290 ENDPROC
4300 :
4310 REM ****************************************
4320 :
4330 DEF PROClimpiar(M%)
4340 LOCAL L%
4350 FOR L%=0 TO 255
4360   tablero%?L%=tablero%?L% AND M%
4370   NEXT
4380 ENDPROC
4390 :
4400 REM ****************************************
```

nes correctas del tablero, correspondiendo cada sentencia a un número de fichas de handicap entre dos y siete.

En el próximo capítulo añadiremos las rutinas que se requieren para jugar de la manera apropiada. Debido a la forma estructurada en que se ha desarrollado el programa, también podremos modificarlo fácilmente para permitir partidas entre dos jugadores, con el ordenador vigilando que nadie haga trampa.

Commodore 64

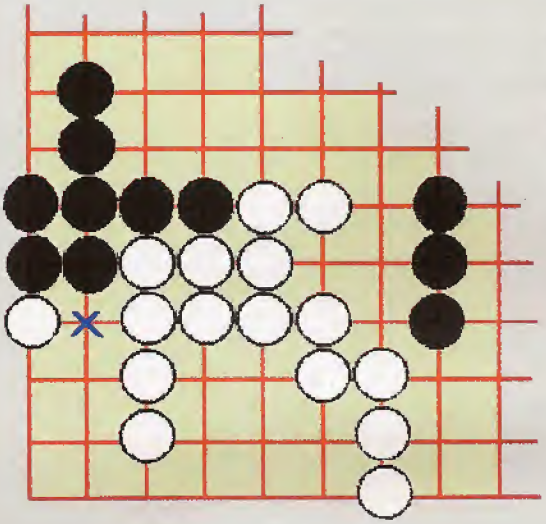
```
 320 GOSUB 600
 590 :
 600 REM RUTINA LEER-HANDICAPS
 620 HNCP=TABLERO+512
 640 FOR L=0 TO 43
 650 READ H%:POKE HNCP+L,H%
 660 NEXT
 670 DATA 68,204
 680 DATA 68,204,76
 690 DATA 68,204,76,196
 700 DATA 68,204,76,196,136
 710 DATA 68,204,76,196,132,140
 720 DATA 68,204,76,196,132,140,136
 730 DATA 68,204,76,196,132,140,72,200
 740 DATA 68,204,76,196,132,140,72,200,136
 750 RETURN
 760 :
 770 REM ****************************************
1400 MSK%=0:GOSUB 4330
1580 GOSUB 1630
1620 :
1630 REM RUTINA HANDICAP
1650 Q%=INT((HND%-2)/2*(HND%+1)+0.5)
1660 FOR L=Q% TO Q%+HND%-1
1670 P%=PEEK(HNCP+L):POKE TABLERO+P%,
       NEGRAS%
1680 NEXT
1690 RETURN
1700 :
1710 REM ****************************************
4030 :
4040 REM RUTINA CONTADOR
4050 CLIB%=0:CSTN%=0
4070 SP%=CP%:SC%=CC%:GOSUB 4130
4080 MSK%=COLOR%:GOSUB 4330
4090 RETURN 0 THEN RETURN
4095 IF (PEEK(TABLERO + SP%) AND COLOR%)=0 GOTO
       4250
4096 IF (PEEK(TABLERO + SP%) AND SC%)=0 THEN
       RETURN
4097 IF (PEEK(TABLERO + SP%) AND MARCADOR%) > 0
       THEN RETURN
4100 :
4110 REM ****************************************
4120 :
4130 REM RUTINA BUSQUEDA
4140 IF (SP% AND 240)<>0 OR (SP% AND 15)<>0
4180 POKE TABLERO + SP%,SC% + MARCADOR%
4190 CSTN=CSTN%+1
4195 SK%(PILA%)=SP%:PILA%=PILA%+1
4200 SP%=SK%(PILA%-1)+DIR%(1):GOSUB
       4130
4210 SP%=SK%(PILA%-1)+DIR%(2):GOSUB
       4130
4220 SP%=SK%(PILA%-1)+DIR%(3):GOSUB
       4130
4230 SP%=SK%(PILA%-1)+DIR%(4):GOSUB
       4130
4235 PILA%=PILA%-1:SP%=SK%(PILA%)
4240 RETURN
4250 IF (PEEK(TABLERO+SP%) AND LICENCIA%)>0 THEN
       RETURN
4260 POKE TABLERO+SP%,LICENCIA%
4270 CLIB%=CLIB%+1
4290 RETURN
4300 :
4310 REM ****************************************
4320 :
4330 REM RUTINA LIMPIEZA
4350 FOR L=0 TO 255
4360 POKE TABLERO+L,PEEK(TABLERO+L)AND
       MSK%
4370 NEXT
4380 RETURN
4390 :
4400 REM ****************************************
```

Amstrad CPC 464/664

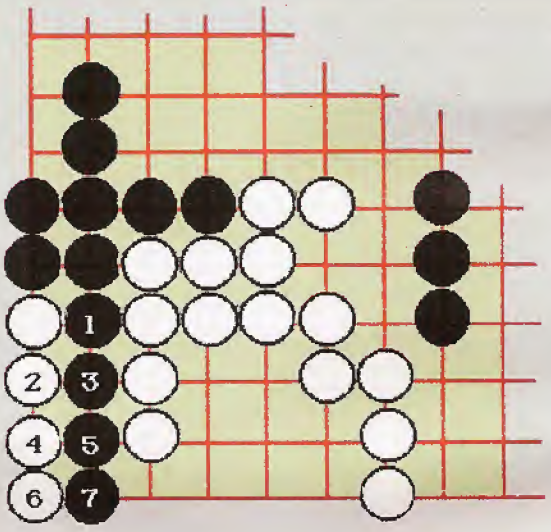
```
 270 GOSUB 600:REM leer handicaps
 590 :
 600 REM rutina leer handicaps
 620 hncp=tablero+&200
 630 RESTORE 670
 640 FOR I%=0 TO 43
 650 READ h%:POKE(hncp+I%),h%
 660 NEXT I%
 670 DATA &44,&cc
 680 DATA &44,&cc,&4c
 690 DATA &44,&cc,&4c,&c4
 700 DATA &44,&cc,&4c,&c4,&88
 710 DATA &44,&cc,&4c,&c4,&84,&8c
 720 DATA &44,&cc,&4c,&c4,&84,&8c,&88
 730 DATA &44,&cc,&4c,&c4,&84,&8c,&48,&c8
 740 DATA &44,&cc,&4c,&c4,&84,&8c,&48,&c8,&88
 750 RETURN
 760 :
 770 REM ****************************************
1400 mascara%=0:GOSUB 4330:REM rutina limpieza
1580 GOSUB 1630:REM rutina handicap
1620 :
1630 REM rutina handicap
1650 q%=INT((hand%-2)/2*(hand%+1)+0.5)
1660 FOR I%=q% TO q%+hand%-1
1670 LET p%=PEEK(hncp+I%):POKE(tablero+p%),
       negras%
1680 NEXT I%
1690 RETURN
1700 :
1710 REM ****************************************
4030 :
4040 REM rutina contador
4050 clib%=0:cstn%=0
4060 cloc%(1)=0:cloc%(2)=0
4070 sp%=cp%:sc%=cc%:GOSUB 4130:REM
       busqueda
4080 mascara%=color%:GOSUB 4330: REM
       limpiar
4090 RETURN
4110 REM ****************************************
4120 :
4130 REM rutina de busqueda
4140 IF (sp%AND 240)=0 OR (sp%AND 15)=0 THEN
       RETURN
4150 IF (PEEK(tablero + sp%)AND color%)=0 THEN
       4250
4160 IF (PEEK(tablero + sp%)AND sc%)= THEN
       RETURN
4170 IF (PEEK(tablero + sp%)AND marcador%)>0 THEN
       RETURN
4180 POKE(tablero + sp%),sc% + marcador%
4190 cstn%=cstn% + 1
4195 s(pila%)=sp%:pila%=pila% + 1
4200 sp%=s(pila%-1)+dir%(1):GOSUB 4130
4210 sp%=s(pila%-1)+dir%(2):GOSUB 4130
4220 sp%=s(pila%-1)+dir%(3):GOSUB 4130
4230 sp%=s(pila%-1)+dir%(4):GOSUB 4130
4235 pila%=pila%-1:sp%=s(pila%)
4240 RETURN
4250 IF (PEEK(tablero + sp%)AND licencia%)>0 THEN
       RETURN
4260 POKE(tablero+sp%),licencia%
4270 clib%=clib%+1
4290 RETURN
4300 :
4310 REM ****************************************
4320 :
4330 REM rutina limpieza
4350 FOR I%=0 TO 255
4360 POKE (tablero+I%),(PEEK(tablero+I%) AND
       mascara%)
4370 NEXT I%
4380 RETURN
4390 :
4400 REM ****************************************
```

Sinclair Spectrum

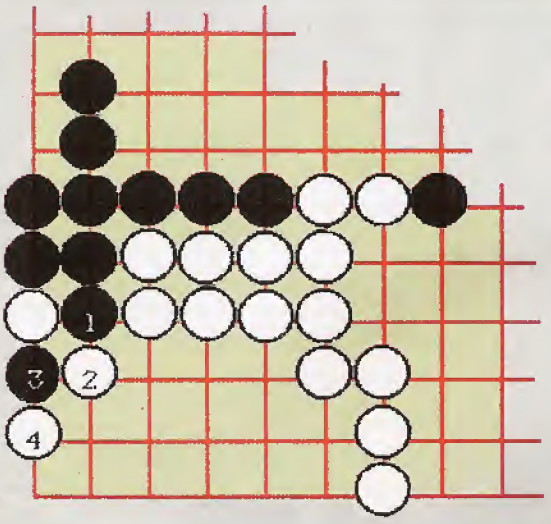
```
 270 GO SUB 600
 590 :
 600 REM rutina leer-handicaps
 620 LET hncp=tablero + 512
 630 RESTORE 670
 640 FOR I=0 TO 43
 650 READ h:POKE hncp+1,h
 660 NEXT I
 670 DATA 68,204
 680 DATA 68,204,76
 690 DATA 68,204,76,196
 700 DATA 68,204,76,196,136
 710 DATA 68,204,76,196,132,
     140
 720 DATA 68,204,76,196,132,
     140,136
 730 DATA 68,204,76,196,132,
     140,72,200
 740 DATA 68,204,76,196,132,
     140,72,200,136
 750 RETURN
 760 :
 770 REM ********************************
1400 LET mascara=0:GO SUB 4330
1580 GO SUB 1630
1620 :
1630 REM rutina handicap
1650 LET q=INT((hand-2)/2* (hand+1)
       +0.5)
1660 FOR I=q TO q+hand-1
1670 LET p=PEEK (hncp+1):POKE tablero+
       p,negras
1680 NEXT I
1690 RETURN
1700 :
1710 REM ********************************
4030 :
4040 REM rutina contador
4050 LET clib=0:LET cstn=0
4070 LET sp=cp:LET sc=cc: GO SUB
     4130
4080 LET mascara=color:GO SUB
     4330
4090 RETURN
4100 :
4110 REM ********************************
4120 :
4130 REM rutina de busqueda
4140 IF INT (sp/16)=0 OR sp-16* INT (sp/16)
       =0 THEN RETURN
4150 IF PEEK (tablero+sp)=licencia OR PEEK
       (tablero+sp)=0 THEN GO TO 4250
4160 IF PEEK (tablero + sp)= color-sc THEN
       RETURN
4170 IF PEEK (tablero + sp)> color THEN
       RETURN
4180 POKE tablero + sp, sc + marcador
4190 LET cstn=cstn + 1
4195 LET s(pila)=sp:LET pila=pila + 1
4200 LET sp=s(pila-1)+d(1): GO SUB
       4130
4230 LET sp=s(pila-1)+d(4): GO SUB
       4130
4235 LET pila=pila-1:LET sp=s(pila)
4240 RETURN
4250 IF PEEK (tablero+sp)>color THEN
       RETURN
4260 POKE tablero + sp,licencia
4270 LET clib=clib + 1
4290 RETURN
4300 :
4310 REM ********************************
4320 :
4330 REM rutina limpieza
4350 FOR I=0 TO 255
4360 IF PEEK (tablero + 1)>mascara THEN POKE
       (tablero +), PEEK (tablero +
       1)-mascara-1:
       GO TO 4360
4370 NEXT I
4380 RETURN
4390 :
4400 REM ********************************
```



Ventaja territorial
En esta situación las blancas han conquistado la esquina inferior izquierda del tablero. Sin embargo, este territorio aún no está completamente a salvo. El punto clave es el señalado mediante la cruz



Poniéndose a cubierto
Si se permite a las negras jugar en este punto clave, este movimiento se denominará un *Atari* en la ficha blanca del borde del tablero Las blancas podrían intentar «correr» hasta la esquina, pero no pueden impedir una ulterior captura por parte de las negras



Maniobra alternativa
En vez de correr hasta la esquina, las blancas podrían sacrificar la ficha situada en el borde del tablero y, de este modo, detener la irrupción de las negras en su territorio jugando como se indica

Caroline Clayton

Crispin Thomas

Para leerlo todo
El Omni-reader, de Oberon, es el primer dispositivo para reconocimiento óptico de caracteres (OCR) a la vez fiable y dentro de la gama de precios para las empresas pequeñas y medianas. Puede leer texto de una página impresa y transferirlo directamente al software de tratamiento de textos de un ordenador a través de una interface en serie RS232C. Esto permite editar el texto en pantalla sin tener que digitar el documento completo. Por el momento, Oberon sólo ha proporcionado software para los micros de gestión más populares, aunque cualquier ordenador con una interface compatible con RS232C adecuada puede utilizar el sistema

Un ojo avizor

Su ordenador, gracias al Omni-reader, podría leer estas mismas palabras que usted está leyendo. Analicemos este interesante dispositivo

En los últimos años se han hecho muchas especulaciones sobre el concepto y el desarrollo de la «oficina sin papeles». La idea proviene del hecho de que, puesto que se pueden transferir datos directamente de un ordenador a otro, en teoría no existe ninguna necesidad de consignar la información en papel. Sin embargo, las predicciones concernientes a esta situación ideal han resultado ser algo prematuras. A la mayoría de nosotros el uso de lápiz y papel sigue resultándonos el método más conveniente para anotar información. Asimismo, durante la fase de «transición» entre la oficina actual y la «oficina sin papeles», los datos aún se deben digitar de forma manual en el ordenador. Para acelerar este proceso, los dispositivos OCR (*optical character recognition:* reconocimiento óptico de caracteres), que aparecieron por primera vez en 1955, son capaces de leer el texto de una página impresa. No obstante, sólo recientemente se han convertido en propuestas prácticas y fiables para su uso en la oficina.

El Omni-reader, de Oberon International, es uno de los nuevos OCR basados en microprocesador que ha aparecido para micros de gestión. Aunque inicialmente el único software disponible es para máquinas tales como el IBM PC y sus compatibles, la gama Apricot y el Apple Macintosh, es factible utilizar el dispositivo con cualquier micro equipado con una interface RS232C adecuada.

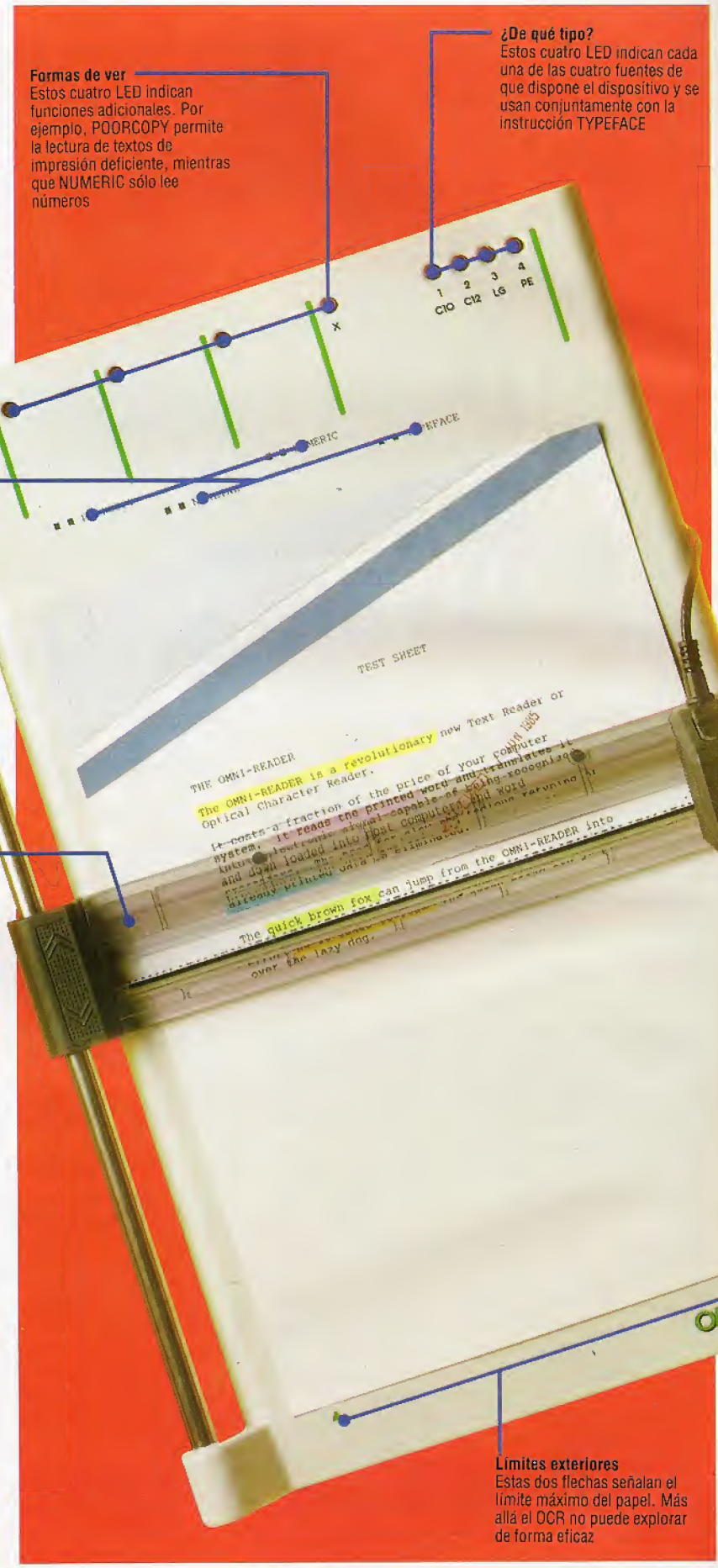
El Omni-reader se compone de una tablilla plástica para documentos con una regla horizontal que se desliza a lo largo de una barra vertical situada a la izquierda de la tablilla. El lector óptico propiamente dicho (un dispositivo manual que tiene dos botones y un LED arriba) está montado en la regla y, por lo tanto, se puede mover a través de un documento colocado sobre la tablilla, de forma muy similar a la de un plotter *xy*.

Se pueden seleccionar numerosas funciones y tipos de letra diferentes, que son indicadas mediante LEDS a lo largo de la parte superior de la tablilla. Actualmente el Omni-reader sólo puede explorar cuatro tipos de letra o «fuentes». Éstos son el Courier 10, Courier 12, Letter Gothic 12 y Prestige Elite 12, que son las cuatro fuentes más populares que se utilizan en las impresoras margarita y las máquinas de escribir electrónicas.

En el interior del lector óptico hay dos fuentes de luz infrarroja a ambos lados de una lente que enfoca la imagen sobre el sistema de circuitos detector de luz. El detector sólo puede leer caracteres impresos con tintas basadas en carbón o toner de fotocopiadoras (que suelen tener una base de carbón), lo que aporta ventajas e inconvenientes. En el lado positivo, significa que las tintas de los bolígrafos y otras tintas no basadas en carbón son «transparentes» para el detector. Por tanto, se puede leer el texto apropiado aun cuando haya sido marcado con

un bolígrafo o un sello de goma. Además, también significa que la mayor parte de los papeles de color no afectarán la operación del Omni-reader. No obstante, las limitaciones del sistema se hacen evidentes por el hecho de que las marcas con lápiz se han de borrar por completo para que el texto se pueda leer correctamente.

El lector óptico lleva un cable que se introduce en la parte posterior de la tablilla del Omni-reader. A lo largo de la parte trasera también se hallan el conector para fuente de alimentación eléctrica, un conector D RS232C estándar de 25 patillas y dos juegos de interruptores DIP, uno para establecer la velocidad en baudios y otro que ofrece funciones de movimiento manual y espaciado de caracteres.



Para leer texto en el ordenador, usted primero debe colocar en la tablilla la copia y después alinear una ventana, marcada en la regla, sobre la línea de texto a leer. Luego se pulsa el botón del lector óptico y la unidad se desplaza a lo largo de la regla, explorando el texto a través de la ventana. Al cabo de aproximadamente un segundo, si el Omni-reader ha leído con éxito la línea, el LED se encenderá una vez, sonará un *beep* y la línea aparecerá en la pantalla. Si el lector no ha leído correctamente la línea, el LED parpadeará y se emitirá el *beep* dos veces. El usuario puede entonces optar por volver a someter la línea efectuando otra pasada a través de la pantalla (pulsando el botón de abajo y volviendo a intentarlo), o bien decir al ordenador que acepte la línea incorrectamente leída pulsando el botón de arriba.

El Omni-reader compara el patrón que lee con la lista de caracteres que posee para ese tipo de letra en particular. Los caracteres fuente se retienen en la memoria del Omni-reader en forma de plantillas de bits, y la unidad obtiene los caracteres explorando constantemente la imagen enfocada a través de la lente. La imagen se divide entonces en 50 «reba-

OMNI-READER

INTERFACES

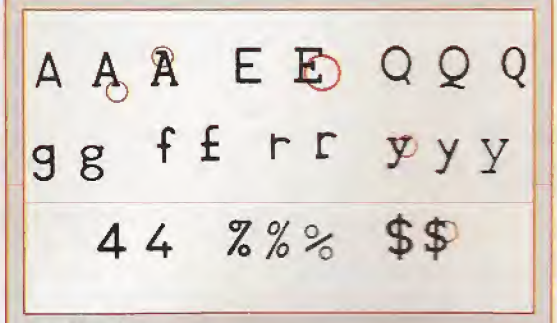
Interface RS232C estándar con velocidad en baudios regulable

VENTAJAS

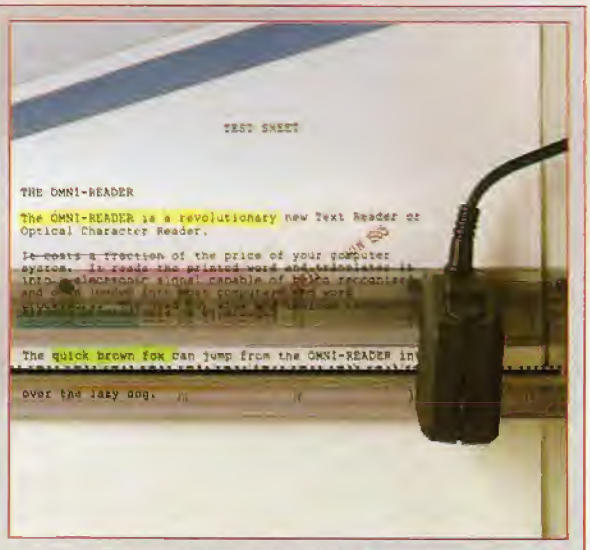
El Omni-reader es veloz y preciso y puede entrar copias en el ordenador de forma mucho más rápida que un mecanógrafo. El dispositivo viene con software y cable de conexión incluidos

DESVENTAJAS

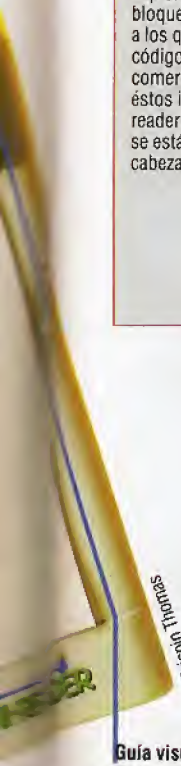
Diversas limitaciones de hardware restringen la lectura precisa a una cantidad limitada de tipos y tamaños de letra. El campo de acción del dispositivo se ve aún más restringido por el hecho de que la tinta haya de tener una base de carbón



Caracteres dudosos
Aunque al observador ocasional le puede parecer que todos los tipos de letra son muy parecidos, existen ciertas diferencias clave que dificultan la detección electrónica de caracteres. Entre éstas, la fundamental son los trazos de pie, extensiones de los caracteres señaladas mediante círculos. Dado que el Omnireader posee plantillas fijas en su memoria, todo lo que no concuerde exactamente con el carácter de la página impresa será rechazado. Sin embargo, aun si pudiéramos proporcionar al carácter alguna clase de «emparejamiento borroso», hay algunas letras, como los ejemplos de la *O* y la *g*, que son muy diferentes



Una ventana a las palabras
Para que el dispositivo de reconocimiento óptico de caracteres trabaje con eficacia, el texto se debe posicionar exactamente en el centro de la ventana. Abajo de la ventana de exploración hay una serie de bloques, bastante similares a los que se observan en los códigos de barras comerciales. Al explorarlos, éstos indican al Omni-reader la dirección en la que se está desplazando el cabezal del OCR



Guía visual
El dispositivo se guía de forma manual a través de la línea de texto. Al final de una línea se enciende el LED superior. Si se enciende una vez, la lectura ha sido correcta; si se enciende dos veces es porque el esfuerzo no ha sido totalmente provechoso. Una luz continua indica que el dispositivo no ha conseguido leer la línea entera

nadas», tanto en dirección horizontal como vertical.

Al desplazarse a través de la línea de texto, el Omni-reader busca caracteres enteros debajo del cabezal. Sólo procesará el carácter que se halle en el centro de su campo de visión y que no esté cortado por ninguno de los límites de la superficie que se esté explorando. (Éste es el motivo por el cual el carácter se explora tanto horizontal como verticalmente.) El carácter se envía luego para cotejarlo con los caracteres de la memoria del dispositivo.

Cuando el Omni-reader encuentra una pareja, el software envía al ordenador el código ASCII equivalente a través de la interface RS232C y el carácter aparece en la pantalla. Debido a que el Omni-reader sólo puede hallar una pareja que concuerde exactamente con la plantilla retenida en la memoria, el texto ha de estar correctamente alineado en relación a la ventana de exploración. Por consiguiente, si las letras con trazo bajo (como las *y, g, p* y *q*, que poseen una parte del carácter por debajo de la «línea») quedan fuera de la ventana, quizá se las tome por otra cosa; por ejemplo, se puede identificar una *p* por una *o*.

Por el contrario, si la ventana se halla demasiado abajo, puede ser que el explorador óptico tome las partes de arriba de los caracteres de la línea de abajo e intente interpretarlos también. Asimismo surgirán problemas si el usuario intenta entrar texto que sea demasiado largo o bien demasiado corto como para caber con comodidad dentro de la superficie de exploración. Oberon aconseja utilizar solamente caracteres de 10 o 12 puntos (cuatro o cinco caracteres por centímetro). Todo cuando quede fuera de esta gama es susceptible de generar un nivel de error inaceptable.

Manteniéndose dentro de estas restricciones, el Omni-reader de hecho trabaja sorprendentemente bien. El tiempo óptimo de exploración para una línea de texto A4 es de entre 0,5 y 1,5 segundos, que es aproximadamente lo que se tarda en realizar una pasada uniforme a través de la línea. Aunque el usuario novel puede experimentar algunas dificultades iniciales para alinear el texto correctamente, aprender a juzgar el espaciado no lleva mucho tiempo; tras media hora ya se pueden producir líneas sin que se produzcan errores.

Para ayudar al usuario a leer los tipos de letra de menor calidad se han añadido algunas opciones. Éstas se suministran al Omni-reader pasando el lector óptico sobre instrucciones escritas; de éstas, cuatro están situadas en la parte superior de la tablilla y el resto están listadas en el manual. Cada instrucción requiere ir precedida por dos cuadrados sólidos para que sea leída como tal y no como una palabra más a visualizar en la pantalla.

De las cuatro instrucciones incluidas en la tablilla la que se utiliza con mayor asiduidad es TYPEFACE, que permite que usted elija la fuente a leer. NUMERIC suprime de la fuente los caracteres alfabéticos y sólo leerá números, acelerando, por tanto, el proceso de lectura. Quizá esta opción sea una de las aplicaciones más útiles del Omni-reader, puesto que generalmente es al copiar cifras cuando se cometen la mayor parte de los errores no detectados.

La entrada directa es un buen método para obviar este problema. Para tratar con las impresiones de menor calidad se puede emplear la opción POORCOPY, que reduce la velocidad de lectura del Omni-reader de modo que pueda obtener una imagen más clara del carácter situado debajo del lector óptico. Resulta de especial utilidad al leer tipos generados mediante sistemas de cinta de tela entintada, en los cuales el carbón es menos denso.

El software Omni-reader se compone de un par de programas activadores que residen en el área de memoria del sistema. Esto significa que al mismo tiempo se pueden cargar en el ordenador programas de aplicaciones. En consecuencia, los datos leídos de un trozo de papel se pueden entrar directamente en un programa de tratamiento de textos como el *WordStar* o, si usted está leyendo cifras, las puede leer sobre las columnas de una hoja electrónica para obtener cálculos instantáneos.

El precio del Omni-reader incluye el software y un cable de conexión para el ordenador que usted elija. El precio también da derecho a medio día de formación y un año de mantenimiento gratuito a domicilio. No se trata en modo alguno de un dispositivo barato. No obstante, allí donde se reciban muchas copias mecanografiadas, como en la redacción de un periódico, invertir en un Omni-reader a la larga será decididamente más económico que emplear un mecanógrafo a tiempo completo.

Pieza clásica

Concluimos esta serie sobre la interface MIDI analizando otras técnicas para escribir software para la misma

En el capítulo anterior examinamos las diferencias existentes entre la grabación analógica normal de una cinta y la grabación de datos digitales MIDI. Vimos también cómo estas diferencias pueden ser aprovechadas para reducir el empleo de la memoria central por medio de mensajes temporizadores.

La grabación digital y la aplicación del playback o ejecución en diferido nos proporciona un buen ejemplo de algunos problemas que comúnmente encontramos cuando se programa para la MIDI. El programa ha de poder recibir y transmitir datos MIDI por medio de la interface hardware diseñada anteriormente; más aún, esos datos deben ser transmitidos y recibidos en tiempo real. En otras palabras, el programa debe ejecutarse lo suficientemente rápido como para aceptar, procesar y almacenar un mensaje MIDI antes de la llegada del siguiente mensaje.

Además, puesto que estamos grabando en tiempo real, debemos disponer de alguna fuente fiable de temporización dentro del ordenador que sincronice las acciones del programa en los modos de grabación y reproducción, así como grabar bytes de temporización dentro de la tabla de memoria empleada para almacenar la secuencia grabada.

Muchas aplicaciones que exigen actualizaciones periódicas están gestionadas con interrupciones, es decir, emplean señales de interrupción IRQ para ejecutar regularmente secciones determinadas de código de programa. Sin embargo, para esta aplicación, el intervalo entre IRQ sucesivas debe ser más largo que el tiempo entre dos mensajes MIDI sucesivos. Cuando el programa está en la opción grabación, esto puede ser catastrófico, dado que todos los mensajes MIDI enteros pueden perderse, con efectos verdaderamente sorprendentes al reproducir la pieza. Por ejemplo, si un mensaje de *«note off»* (sonido excluido) se perdiera durante la fase de grabación, esa nota o sonido, una vez generada, estaría sonando indefinidamente. Debemos, pues, desactivar las interrupciones IRQ y buscar otra fuente de temporización para nuestro programa.

Temporización para MIDI

Los chips de E/S que emplean el Commodore 64 y el BBC Micro son parecidos y cada uno tiene un registro temporizador de 16 bits incorporado, que puede ser programado directamente para que cuente hacia atrás, desde un determinado valor hasta cero. Si opera así, se dice que el temporizador está funcionando en *modo de libre ejecución*.

El programa utiliza este temporizador como sigue: la subrutina check comprueba si el temporizador se ha «agotado» interrogando al registro flag IRQ del chip CIA/VIA y pone el flag Z en el registro de estado del procesador en el modo conveniente. Las secciones de grabación y reproducción del programa llaman, ambos, a la subrutina check y emplean las instrucciones BNE y BEQ al volver de esta rutina, bifurcando según el estado del flag Z.

La rutina check inspecciona también el teclado para ver si se ha pulsado la barra espaciadora para detener la secuencia. En el Commodore 64 esta inspección se realiza, en ausencia del rastreador de teclado normal gestionado por IRQ, comprobando la última fila de teclas en el teclado. La versión del BBC Micro comprueba el teclado a través de la llamada &79 OSBYTE. Siempre que hagamos que esta rutina check sea llamada al menos una vez entre dos finales de tiempo sucesivos, tendremos un método fiable para temporizar la recepción y transmisión de mensajes MIDI.

No es necesario, ni siquiera recomendable, grabar todo mensaje enviado en MIDI. Dado que el programa *Grabación/reproducción* genera su propia temporización, los mensajes de sistema en tiempo real son irrelevantes y deben ser ignorados si son recibidos. (Muchos instrumentos de teclado son incapaces de transmitir estos códigos sin un secuenciador incorporado.)

Los mensajes de sistema comunes y exclusivos pueden ser emitidos inadvertidamente y deben ser ignorados. Sin embargo, estos mensajes pueden componerse de cualquier número de bytes y por ello se ha de utilizar un flag para instruir a la grabadora de que ignore todos los datos hasta que no se reciba un byte de estado de canal. Esto lo indica el valor $FF en el registro X. Otros valores contenidos en el registro X actúan como contadores que representan el número de bytes de datos que se han de recibir antes de que se complete el mensaje en curso del canal.

Posibles mejoras

Este sencillo programa es esencialmente un ejemplo del tipo de problemas que se encuentran cuando se procesan datos MIDI en tiempo real. Varias son las posibles mejoras que pueden hacerse al programa, que exigen diferentes esfuerzos de programación.

1. Si se detiene la reproducción en una semisecuencia dejará generalmente sonando un número indefinido de notas dado que no se han recibido mensajes de «exclusión de notas».
2. No se ha previsto la sincronización de otros dispositivos tales como máquinas de ritmos y grabadoras de cinta.
3. No es posible editar la secuencia.
4. El tempo de grabación y reproducción es fijo.
5. No se puede silenciar una determinada secuencia.
6. No es posible grabar mientras se visualiza una secuencia diferente.
7. Si el cable MIDI se desconecta durante el mensaje, el programa quedará colgado.
8. No se ha previsto el guardado y la carga de las secuencias grabadas en disco o cassette.

Prog. «Grabación/reproducción»

```
  5 REM **   PROYECTO MIDI EN BBC
  6 REM **   GRABACION DIGITAL
  7 REM **   Y REPRODUCCION
  8 *FX 14,6
 10 DIM start &2000
 20 FOR op t%=0 TO 3 STEP 3
 30 timer=&FE60
 40 osasci=&FFE3
 50 osrdch=&FFE0
 60  osbyte=&FFF4
 70 streg=&FEE0
 80 datreg=streg+1
 90 mem=&70
100 ptr=&72
110 P%=start
120 [
130  OPT opt%
140 .begin
150 JMP start1
160  .nobytes BRK
170  .clocks BRK
180  .freemem BRK
190  BRK
200 .start1
210 LDA #3
220 STA streg
230 LDA #&16
240 STA streg
250 .g10
260 LDA #&FF
270 STA 1 owmem
280 LDA #&40
290 STA timer+11
300 LDA #0
310 STA timer+4
320 LDA #&08
330 STA timer+5
340 .g20
350 LDA #0
360 JSR strout
370 .g21
380 JSR osrdch
390 CMP #ASC"E"
400 BNE g22
410 RTS
420 .g22
430  CMP #ASC"R"
440 BEQ g30
450 CMP #ASC"P"
460 BNE g21
470 .g30
480 PHA
490 LDA #1 owmem MOD 256
500 STA mem
510 LDA #1 owmem DIV 256
520 STA mem+1
530 LDA #frbytes MOD 256
540 STA freemem
550 LDA #frbytes DIV 256
560 STA freemem+1
570 SEI
580 LDY #0
590 PLA
600 CMP #ASC"R"
610 PHP
620  BEQ r00
630 .p00
640 LDA #2
650 JSR strout
660 ..p05
670 JSR read
680 CMP #&FF
690 BNE p07
700 PLP
710 JMP c20
720 .p07
730 STA clocks
740 CMP #&F0
750 LDA clocks
760 BEQ p30
770 .p10
780 JSR check
790 BEQ p10
800 DEC clocks
810 BNE p10
820 BCS p05
830 .p30
840 JSR read
850 PHA
860 .p35
870 LDA streg
880 AND #2
890 BEQ p35
900 PLA
910 STA datreg
920 BPL p50
930 JSR getno
940 .p50
950 DEX
960 BNE p30
970 LDX nobytes
980 BPL p05
990 .r00
1000 LDA #1
1010 JSR strout
1020 STY clocks
1030 .r05
1040 LDX #&FF
1050 .r10
1060 JSR check
1070 BEQ r20
1080 INC clocks
1090 LDA clocks
1100 CMP #&F0
1110 BCC r20
1120 JSR store
1130 STY clocks
1140 .r20
1150 LDA streg
1160 AND #1
1170 BNE r40
1180 CPX #1
1190 BMI r10
1200 BPL r20
1210 .r40
1220 LDA datreg
1230 BMI r50
1240 CPX #0
1250 BMI r10
1260 BNE r80
1270 LDX nobytes
1280 BPL r60
1290 .r50
1300  CMP #&F8
1310  BCS r20
1320  CMP #&F0
1330 BCS r05
1340 JSR getno
1350 .r60
1360 PHA
1370 LDA clocks
1380 JSR store
1390 STY clocks
1400 PLA
1410 .r80
1420 JSR store
1430 DEX
1440 BEQ r10
1450 BNE r20
1460 .check
1470 STX &74
1480 PHP
1490 LDA #&79
1500 LDX #&E2
1510 JSR osbyte
1520 LDY #0
1530 PLP
1540 TXA
1550 BPL c40
1560 PLA
1570 PLA
1580 PLP
1590 BNE c20
1600 LDA #&FF
1610 STA (mem),Y
1620  .c20
1630 CLI
1640 JMP g20
1650 .c40
1660 LDX &74
1670 LDA timer+13
1680 AND #&40
1690 STA timer+13
1700 RTS
1710 .store
1720 STA (mem),Y
1730 INC freemem
1740 BNE point
1750 INC freemem+1
1760 BNE point
1770 PLA
1780 PLA
1790 PLA
1800 LDA #3
1810 JSR strout
1820 JMP c20
1830 .read
1840 LDA (mem),Y
1850 .point
1860 INC mem
1870 BNE p20
1880 INC mem+1
1890  .p20 RTS
1900 .getno
1910 LDX #2
1920 CMP #&C0
1930 BCC n10
1940 CMP #&E0
1950 BCS n10
1960 DEX
1970 .n10
1980 STX nobytes
1990 INX
2000 RTS
2010 .strout
2020 ASL A
2030 TAX
2040 LDA messtab,X
2050 STA ptr
2060 LDA messtab+1,X
2070 STA ptr+1
2080 LDY #0
2090 .m10
2100 LDA (ptr),Y
2110 JSR osasci
2120 INY
2130 CMP #&0D
2140 BNE m10
2150 LDY #0
2160 .m70 RTS
2170 ]
2180 mess0=P%
2190 $P%="R = grabar,P = reproducir E = salir"
2200 P%=P%+LEN($P%)+1
2210 mess1=P%
2220 $P%="grabacion"
2230 P%=P%+LEN($P%)+1
2240 mess2=P%
2250 $P%="reproducción"
2260 P%=P%+LEN($P%)+1
2270 mess3=P%
2280 $P%="no queda memoria"
2290 P%=P%+LEN($P%)+1
2300 1 owmem=P%+8
2310 frbytes=&E000+(1 owmem-start)
2320 [
2330 .messtab
2340 ]
2350 NEXT
2360 ?P%=mess0 MOD 256
2370 P%?1=mess0 DIV 256
2380 P%?2=mess1 MOD 256
2390 P%?3=mess1 DIV 256
2400 P%?4=mess2 MOD 256
2410 P%?5=mess2 DIV 256
2420 P%?6=mess3 MOD 256
2430 P%?7=mess3 DIV 256
2440 CALL begin
```

Tortuga pequeña

Compararemos la versión Amstrad de Dr LOGO con la producida para el IBM PC

Cuando se transfirió por primera vez el LOGO desde ordenadores centrales a microordenadores de ocho bits, hubieron de introducirse varias restricciones y simplificaciones con el fin de que se pudiera ejecutar el lenguaje con la memoria disponible, de 64 K o incluso menos. Cuando comenzaron a aparecer micros de 16 bits, con memorias mucho mayores, enseguida salieron al mercado numerosas versiones mejoradas de LOGO.

Gary Kildall (el fundador de Digital Research, productora del sistema operativo CP/M) se sintió favorablemente impresionado por el potencial del LOGO como lenguaje y produjo su propia versión para el IBM PC, a la que llamó Dr LOGO. Desde entonces esta versión se ha puesto al alcance de otras varias máquinas de 16 bits. El Apricot F1e, por ejemplo, se suministra ya con Dr LOGO.

Desde entonces Digital Research ha adaptado Dr LOGO para numerosas máquinas CP/M de ocho bits. Amstrad está proporcionando esta versión de Dr LOGO junto con su paquete de disco para el CPC 464.

Dr LOGO, en su implementación para el IBM PC, requiere para su ejecución al menos 192 K de memoria. El disco no se puede copiar porque está protegido, pero se proporciona una copia de seguridad. Para ejecutarlo, se debe insertar el disco y efectuar una inicialización del sistema. La resolución de gráficos en color es de 320 por 200 pixels, con una selección de 16 colores de fondo, junto con una opción de cuatro juegos de tres colores de lápiz (primer plano).

Con una placa de gráficos a color en el PC, junto con una pantalla tanto en color como monocromática, Dr LOGO puede producir gráficos en la pantalla en color y texto en la pantalla monocromática. No obstante, si usted posee una sola pantalla, se pueden mezclar ambos en ella.

En el manual se describe a Dr LOGO como una ampliación del LOGO Apple y en realidad contiene todas las instrucciones de LOGO que contiene aquél. Éstas incluyen las instrucciones estándares para gráficos, proceso de listas y gestión del espacio de trabajo, así como primitivas para tratamiento de errores, «propiedades» (una forma de asignarle a un objeto más de un valor) y «paquetes» (una forma de empaquetar procedimientos entre sí en forma de grupo). Dr LOGO es «sensible a los tipos de letra» y, por tanto, las instrucciones del LOGO se deben entrar en minúsculas.

El editor hace uso de las teclas de función IBM, pero también reconoce los mismos códigos de control utilizados en el LOGO Apple. Si se encuentra un error durante la ejecución de un programa, digitan-

Comparando

PROCEDIMIENTO		RESULTADOS			
		IBM PC (256 K)	AMSTRAD CPC 464/664	COMMODORE 64	APRICOT F1e
RECUR	Nivel alcanzado	512	187	90	832
DEFVARS		1307	263	553	1606
ENDRECUR1		512	189	INFINITO	INFINITO
ENDRECUR 2		151	129	65	72
LINES (con tortuga escondida)	Tiempo (al segundo más próximo)	9	53	13	10
LINES (con tortuga escondida)		5	39	11	6
ASSIGN		25	22	6	20
LISTS		64	74	18	69
ARITH		80	138	36	77
RECURA		92	91	24	72

Procedimientos comparativos

Los nueve procedimientos de comparación están diseñados para probar diferentes características del LOGO y para mostrar los puntos fuertes y los puntos débiles de cada implementación.

- RECUR mide el tamaño de la pila
- DEFVARS mide el espacio disponible para definir variables y procedimientos
- ENDRECUR1/ENDRECUR2 determinan si se ha implementado eficazmente la recursión final. Una implementación ideal permitiría la recursión infinita. Mientras que ENDRECUR2 produce su resultado y, en consecuencia, se puede considerar como una operación, ENDRECUR1 es una instrucción
- LINES mide la velocidad de dibujo. Se dan dos cifras para indicar la mejora que se obtiene en cuanto a velocidad al esconder la tortuga
- ASSIGN mide la velocidad a la cual se pueden asignar valores a las variables
- LISTS mide la velocidad de los operadores para proceso de listas
- ARITH mide la velocidad de las funciones aritméticas
- RECURA mide la vel. de las llamadas recursivas

do *ed* se llamará al editor y éste aparecerá ya con el procedimiento que generó el error, listo para editarlo. Lamentablemente, no hay ninguna forma de conocer el punto exacto del error mirando a la pantalla cuando uno se halla en la modalidad de edición. La mayoría de las otras versiones del LOGO indican esto mediante la inclusión de una línea especial en la parte inferior de la pantalla.

Además de las primitivas para proceso de listas habituales, Dr LOGO posee algunas otras nuevas:

sort clasifica una lista por orden alfabético
shuffle ordena de forma aleatoria los elementos de una lista
piece permite seleccionar una parte de una lista

Dr LOGO añade otras para depuración:

watch permite ver el efecto del procedimiento ejecutándolo línea a línea
trace imprime los nombres de las variables a medida que se las va llamando y definiendo
debug divide la pantalla en dos ventanas. La ventana *debug* se puede utilizar para visualizar información proporcionada por "trace" y "watch", mientras que la ventana *program* muestra la salida del programa

A los procedimientos se les puede añadir comentarios, que después se pueden suprimir de todos los procedimientos en el espacio de trabajo, mediante noformat, si así fuera necesario para proporcionar espacio extra para la ejecución.

Existen, asimismo, algunas primitivas nuevas para administración de procedimientos:

follows permite definir el orden por el cual se deben presentar los procedimientos en la pantalla
potl visualiza nombres de procedimientos no llamados desde ningún otro procedimiento
poref< nombre> visualiza nombres de procedimientos que llaman al procedimiento <nombre>
pocall < nombre> visualiza nombres de procedimientos llamados por el procedimiento <nombre>

Con estas facilidades adicionales se pretende ofrecer un entorno mejorado para el desarrollo de programas, en el cual crear programas en LOGO.

Dr LOGO posee algunos puntos débiles: el editor no posee ninguna función de búsqueda ni de sustitución, no hay ningún método de acceder al código máquina y ninguna gestión de archivos. Sin embargo, el manual de 300 páginas es muy claro y exhaustivo. Posee una introducción al LOGO, así como un manual de referencia que incluye una página separada dedicada a cada primitiva.

Evidentemente, Dr LOGO posee numerosas características extras inexistentes en las versiones más pequeñas del lenguaje, todo lo cual crea un entorno para desarrollo de programas muy mejorado.

Es interesante comparar Dr LOGO, desde el punto de vista de espacio y velocidad, con las versiones de ocho bits del lenguaje. El espacio de trabajo del LOGO se mide por *nudos*. Si se le pregunta a Dr LOGO cuántos nudos libres tiene al arrancar el sistema, la alentadora respuesta será de alrededor de 10 000 (típicamente, los sistemas de ocho bits poseen entre 2 000 y 3 000). De modo, pues, que hay muchísimo espacio de trabajo para definir procedimientos y para ejecutar procedimientos recursivos. No obstante, la recursión final (caso de un procedimiento recursivo en el cual la llamada recursiva se halla en la última línea) no está implementada eficazmente, y, en consecuencia, programas que en las máquinas Commodore o BBC se ejecutarían hasta el infinito, en Dr LOGO se quedan sin espacio luego de pocos cientos de llamadas recursivas.

En cuanto a velocidad, los gráficos son rápidos, pero la aritmética y el proceso de listas se llevan a cabo más lentamente que con otras versiones de ocho bits del lenguaje.

La versión Amstrad

Dr LOGO se suministra con la unidad de disco Amstrad y se halla en la cara opuesta del disco CP/M. El Dr LOGO del Amstrad está modificado para sacar partido del hardware en el que se ejecuta. Los gráficos utilizan el mismo sistema de cuatro lápices que el Dr LOGO para el IBM PC, pero en este caso los colores del lápiz se establecen dando una lista de números que determinan las proporciones de rojo, verde y azul requeridas. La versión IBM no posee una amplia gama de instrucciones de sonido, mientras que la versión para el Amstrad posee env (envoltura de volumen), ent (envoltura de tono) y release, que libera canales establecidos en estado de sostenimiento en una instrucción de sonido.

La versión Amstrad posee la mayoría de las facilidades del LOGO Apple, incluyendo propiedades y tratamiento de errores, pero no paquetes.

Las primitivas de depuración y las facilidades extras para administración de procedimientos del Dr LOGO del IBM PC no se han incluido, presumiblemente, debido a la menor capacidad de memoria del Amstrad. Más grave, no obstante, es el hecho de que no haya ninguna función define y aparentemente ninguna forma de suprimir un archivo sin salir al CP/M y luego volver otra vez.

La capacidad de respuesta del editor es muy lenta y puede ser que el usuario se encuentre digitando por adelantado y perdiendo el carácter suelto. Por otra parte, si se encuentra un error al ejecutar un procedimiento, el editor, una vez entrado, posiciona el cursor sobre la palabra que generó el error en el procedimiento, lo que realmente es una característica muy útil.

La versión Amstrad proporciona un acceso limitado al código máquina a través de .examine y .deposit (PEEK y POKE), de las que carece la versión para el IBM PC.

Se entrega una documentación mínima, remitiendo al usuario a un manual de formación y de referencia que aún no se ha publicado. Mientras tanto, éste tendrá que contentarse con el resumen de 25 páginas de las instrucciones. Asimismo, parece haber algunas facilidades de Dr LOGO que se han implementado pero que no se mencionan.

En la versión para el Amstrad han desaparecido muchísimas de las características más interesantes del Dr LOGO para el IBM PC, a pesar de lo cual continúa siendo una buena implementación estándar de LOGO, con extras tales como empaquetamiento y tratamiento de errores.

Tras el encendido, se afirma que Dr LOGO posee 2 105 nudos, cifra típica para los sistemas de ocho bits. Sin embargo, la recursión final no se ha implementado eficazmente; y en cuanto a gráficos, aritmética y proceso de listas, éste es uno de los LOGO de ocho bits más lentos, consumiendo la mayoría de las operaciones alrededor de tres veces más tiempo que en el Commodore 64.

Introducción

Iniciamos esta serie dedicada al FORTH examinando dos características clave de este lenguaje: la interactividad y la ampliabilidad

Inventado a finales de los años sesenta por el astrónomo Charles Moore, el FORTH se diseñó específicamente para controlar un dispositivo científico que necesitaba un posicionamiento de precisión. La utilidad del lenguaje va, sin embargo, mucho más allá de las aplicaciones de control, pero a modo de introducción a sus principios operativos básicos es interesante observar cómo puede ser utilizado para controlar un telescopio. Sería gratificante sentarse en el observatorio frente a un terminal de ordenador y, por ejemplo, digitar:

```
30 GRADOS ELEVACION
```

y el telescopio se moviera inmediatamente y con toda precisión a dicho punto. También sería práctico poder empaquetar y almacenar secuencias útiles de instrucciones para economizar la digitación repetitiva. Por ejemplo, si la posición de estacionamiento fuera de 90° de elevación y 0° de acimut (términos que describen el ángulo y la posición del telescopio), usted podría describir una nueva instrucción, PARK, como:

```
0 GRADOS ACIMUT 90 GRADOS ELEVACION
```

En resumen, usted necesitaría:

- Instrucciones elementales para el telescopio.
- Formas de combinar las instrucciones elementales en otras más complejas: «programas» para telescopio.

Este sistema equivaldría a un lenguaje de programación completo para telescopios. Sin embargo, requeriría una propiedad extra, bastante especial: habría de aceptar instrucciones (tanto las incorporadas como ELEVACION y otras nuevas) directamente desde el teclado. Además, habría de hacer todo esto de la manera más simple posible.

Intente imaginar lo que sería satisfacer estas exigencias en BASIC, que es bastante bueno en cuanto a permitir que usted utilice sus instrucciones incorporadas, como PRINT y RUN, desde el teclado. Las instrucciones nuevas, sin embargo, serán un problema. Suponiendo que ya hubiera definido ACIMUT y ELEVACION como rutinas GOSUB en las líneas 1000 y 1100, podría escribir una instrucción para PARK como:

```
1200 REM PARK
1205 LET GRADOS=0
1210 GOSUB 1000
1215 LET GRADOS=90
1220 GOSUB 1100
1225 RETURN
```

O quizá pudiera predefinir una variable PARK mediante:

```
LET PARK = 1200
```

y posteriormente impartir la instrucción (en algunas versiones de BASIC):

```
GOSUB PARK
```

Probablemente lo mejor que podría hacer usted fuera definir la rutina GOSUB como un procedimiento (como en el BASIC BBC) y digitar algo como:

```
PROCPARK
```

Por tanto, siempre que llame a una de sus propias rutinas de telescopio, debe entrar instrucciones adicionales como GOSUB o PROC así como el nombre o el número de línea de su rutina.

Una alternativa es escribir un intérprete, un programa que tome instrucciones para el telescopio y las ejecute, algo así como:

```
100 INPUT C$
150 IF C$ = "PARK" GOSUB 1200:GOTO 100
```

Pero entonces usted ya no podrá utilizar las instrucciones del BASIC (como PRINT) de forma directa.

Estamos ahora en condiciones de apreciar que las dos propiedades de nuestro lenguaje para telescopio serán:

- En primer lugar, ha de ser *interactivo*. Debe ser capaz de ejecutar sus instrucciones apenas usted las digite. Esto es como en BASIC, LOGO, APL y PROLOG, pero no como en PASCAL, ALGOL, FORTRAN y COBOL. También algo parecido al módulo de interpretación de comandos de algunos sistemas operativos, como el CP/M y el Unix.
- En segundo lugar, ha de ser *ampliable*. El forma-

Ian McKinnell

Para iniciarse en el lenguaje
El Jupiter Ace fue un valiente intento por comercializar una máquina que como lenguaje estándar ofreciera FORTH en lugar de BASIC. Lamentablemente, la falta de facilidades de color en el Ace, junto con una pequeña memoria, el desconocimiento del FORTH y la fuerte competencia del Sinclair Spectrum lo colocaron en una situación de evidente desventaja respecto a otros micros de su categoría. Sin embargo, una máquina de segunda mano junto con la documentación original (que era de un nivel muy elevado) proporcionarían una excelente introducción al lenguaje

to que haya definido usted para llamar a las instrucciones debe ser idéntico al empleado para las instrucciones incorporadas. Luego, tras haber definido sus propias rutinas, será como si usted estuviera utilizando un lenguaje ampliado, más poderoso, y no el lenguaje original más algunas rutinas nuevas. Esto es como en LOGO y los sistemas operativos, pero no como en ninguno del resto de los lenguajes.

El concepto de ampliabilidad en realidad es aún más importante de lo que pueda desprenderse de lo dicho hasta ahora. Si, por ejemplo, alguien nos vende un lenguaje para control de telescopios y nosotros le añadimos pequeñas ampliaciones para que incluya rutinas útiles para nuestro propio telescopio, habremos ampliado un «lenguaje para control de telescopios de propósito general» original, convirtiéndolo en un nuevo «lenguaje personalizado para control de telescopios». Pero para comprender cabalmente el concepto de ampliabilidad, deberíamos poder empezar con un «lenguaje de ordenador ampliable de propósito general» y ampliarlo hasta convertirlo en un «lenguaje para control de telescopios de propósito general» (en realidad, el mismo se podría ampliar a lenguajes para controlar robots, experimentos científicos, cadenas de producción o cualquier dispositivo que se pueda conectar a un ordenador). Además, no es necesario que lo que se controle sea físico. Podría ser un concepto abstracto interno del ordenador, como la tortuga en una pantalla de gráficos tortuga, los registros en un sistema de archivo o, de hecho, cualquier cosa que usted deseara manipular por medio de un programa.

De los enfoques de que disponemos, el FORTH es el que más se aproxima a un «lenguaje de ordenador ampliable de propósito general». Hemos hablado de los problemas especiales (interactividad y ampliabilidad) que el FORTH intenta resolver, y en nuestro próximo capítulo comenzaremos a hablar de sus soluciones.

Dialectos de FORTH

Existen tres dialectos principales de FORTH: fig (FORTH Interest Group) FORTH, FORTH-79 y FORTH-83. La mayoría de las implementaciones se describen de acuerdo a alguna de estas tres. El FORTH-79 y el FORTH-83 son estándares sucesivos y especifican los juegos mínimos de instrucciones y rutinas que han de estar disponibles. Casi con toda seguridad las implementaciones individuales le ofrecerán más, pero si usted escribe un programa ateniéndose estrictamente al FORTH-83, por ejemplo, lo podrá ejecutar en todas las implementaciones FORTH-83, aun cuando no haga un uso completo de todas las facilidades disponibles. El FORTH-83 es el estándar actual y nos atendremos a él. El FORTH-79 en líneas generales es igual, con muy pocas incompatibilidades. También posee numerosas facilidades extras, incorporadas al construirse la implementación, que sus usuarios tienden a considerar como estándares. Muchas implementaciones del FORTH-79 y el FORTH-83 son versiones del figFORTH modificadas para adecuarlas al estándar

El telescopio y la tortuga

El FORTH y el LOGO son lenguajes muy similares. Ambos se diseñaron originalmente para controlar objetos físicos: un telescopio y una tortuga. Además tienen en común dos conceptos operativos fundamentales: una rutina se llama simplemente entrando su nombre, y se definen rutinas nuevas utilizando rutinas ya existentes. La principal diferencia entre los dos lenguajes proviene de las distintas expectativas de los usuarios a los que están dirigidos. Los usuarios de FORTH exigen velocidad de ejecución (de modo que el lenguaje no sea sólo interactivo y ampliable sino también eficaz) y para conseguirlo están dispuestos a admitir algunas incomodidades. El LOGO, por otra parte, está dirigido básicamente a ser utilizado por niños en edad escolar y no permite tales concesiones. En consecuencia, el LOGO opera con mucha más lentitud. Comparemos los dos lenguajes en mayor profundidad. El FORTH no tiene incorporados gráficos de tortuga como estándar, pero posee facilidades para manipular la pantalla terminal. Usted puede utilizar estas dos facilidades de bajo nivel para definir las palabras de tortuga FORWARD, RIGHT, etc., creando, por tanto, una versión «amplia» del FORTH que incluya gráficos de tortuga. Consideremos estas formas alternativas de controlar a la tortuga en los dos lenguajes:

LOGO	**FORTH**
FORWARD 100	100 FORWARD

A partir de este ejemplo, podría parecer que el FORTH hace las cosas al revés. Pero cuando una rutina en FORTH necesita parámetros, usted primero debe calcular estos parámetros y escribirlos antes del nombre de la rutina. Esto se podría considerar como una técnica de «libro de recetas»: primero reunir los ingredientes (en cantidades especificadas) y luego cocinarlos. Otras correspondencias con el LOGO incluyen la definición de rutinas nuevas:

```
TO...END                    :...;
```

y estructuras de bucle:

```
REPEAT 4[ ..]               4 DO ..LOOP
```

He aquí cómo podríamos combinar las dos en la definición de un procedimiento CUADRADO:

```
TO CUADRADO LADO            :CUADRADO
REPEAT 4[                   4 DO
  FORWARD LADO                DUP ADELANTE
  RIGHT 90                    90 DERECHA
]                           LOOP
                            DROP

END                         ;

CUADRADO 100                100 CUADRADO
```

DUP y DROP son «manipulaciones de pila» que permiten que las rutinas accedan al parámetro de CUADRADO (DUP la reproduce, haciendo una copia para ADELANTE;DROP la «abandona»; en otras palabras, la elimina de modo que no se cruce en el camino del siguiente cálculo)

Marcus Wilson-Smith

De especial interés...
Existen implementaciones de FORTH para muchos micros personales. Entre los paquetes más accesibles se incluyen: FORTH Acornsoft (también disponible en ROM) y SkyWays MultiFORTH (con multitareas) para el BBC Micro, y Abersoft FORTH y Artic FORTH para el Sinclair Spectrum. El FORTH es el único lenguaje de ordenador que ha sido difundido por sus usuarios (que originalmente se enviaban implementaciones a través de Clubes de Usuarios). En Londres existe un floreciente Club de Usuarios de FORTH. El grupo se reúne a las 7 de la tarde el primer jueves de cada mes en el South Bank Polytechnic, Borough Road, London SE1. Usted puede asociarse (la suscripción anual es de £ 7) escribiendo a:
FORTH Interest Group
7 Wyndham Cresent
Woodleigh
Reading
RG5 3AY7719
Gran Bretaña

Todo un serial

El análisis general que hemos llevado a cabo de las rutinas de la Interface 1 exige que nos ocupemos ahora de los códigos de enganche con los que controla la puerta serial RS232

Comenzaremos por dar una visión general de una interface serial, con un enchufe RS232 de 270° y nueve patillas (el dibujo ilustra la función de cada patilla). El número de estas líneas necesarias para la conexión depende de la complejidad de la aplicación, pero un enlace mínimo puede establecerse con las patillas 2, 3 y 7. Si usted hace esto las patillas 4 y 5 han de estar conectadas a la patilla 9 para mantenerlas en 1 lógico. Esto evita que el Spectrum quede colgado si, estando lo que sea en el otro extremo, el enlace todavía no se halla preparado para recibir datos desde el ordenador. Esto tiene la desventaja, sin embargo, de que se pierden los datos si el otro extremo del enlace no está disponible para recibirlos. En general, es preferible el empleo del cableado completo.

Una aplicación posible del enlace RS232 es la conexión del Spectrum a una impresora *real*. Una vez desplegado el cable, el software de control muestra un manejo muy sencillo. Para enviar texto a la impresora, establecemos la velocidad en baudios desde el BASIC por medio de FORMAT, y ejecutamos después el siguiente código:

```
CLOSE #3
OPEN #3;"T"
```

Esto acopla la corriente 3, que es la corriente usual de impresora en el Spectrum, al canal T, que es el canal de texto del enlace RS232. El manual de la Interface 1 da más detalles sobre el empleo de las corrientes *texto* y *binaria*. LLIST y LPRINT enviarán ahora los datos al dispositivo ajustado a la Interface 1.

Veamos ahora algunos detalles de cómo opera la interface serial. Los datos se transmiten con un bit de inicio, ocho bits de datos y dos bits de parada. No hay por qué preocuparse por este formato, dado que las rutinas del sistema operativo del Spectrum ya se encargan de ello. Lo que más importa es la velocidad en baudios a la que son enviados los datos. Esta velocidad suele establecerse con la instrucción FORMAT, que nos limita a una gama determinada de velocidades. Aunque hay poca pérdida de la integridad de los datos con una interface RS232, incluso a considerable distancia, siempre es bueno emplear velocidades más bajas de transmisión según aumenta la distancia entre el emisor y el receptor.

Es un procedimiento recomendable insertar variables de sistema de la Interface empleando el código de enganche 49 antes de usar cualquiera de las siguientes técnicas. Hay dos códigos de enganche referidos al RS232 y tres variables de sistema. Veamos primero las variables.

- BAUD (posiciones 23747 y 23748) es una variable de dos bytes que contiene un valor que determina la velocidad en baudios que va a emplearse en la transmisión y recepción de datos. Esto representa una de las limitaciones de la interface del Spectrum: su incapacidad de manejar dos velocidades de transmisión distintas para enviar y para recibir.
- SERFL (posición 23751) es un byte flag, empleado por el OS sólo al recibir. Si está a uno, quiere decir que hay todavía un byte de lectura disponible.
- SERBYT (posición 23752) actúa como un buffer de entrada de un byte en recepción. Ocasionalmente es posible recibir y almacenar un byte en esta posición después de que usted haya cortado la comunicación. Pueden presentarse problemas con el empleo de las dos últimas variables, como veremos.

Una cuarta variable de sistema, la IOBORD (posición 23750) no se refiere estrictamente a la interface serial, sólo contiene el color del marco de la pantalla (*border*) que se emplea en las operaciones de E/S. Por tanto, puede hacer que el marco contenga cualquier color durante estas operaciones.

Cuando queremos emplear la interface RS232, lo primero que hay que hacer es establecer la velocidad de transmisión. Esto se hace dando un valor apropiado a BAUD, que, en realidad, es todo lo que hace FORMAT. El valor a usar para una determinada velocidad se calcula así:

Velocidades baudio con la Interface 1

La RS232 estándar se emplea con profusión en las comunicaciones, en especial cuando la transmisión de datos se realiza a larga distancia, y la *velocidad baudio* proporciona una indicación del número de bits de datos por segundo que se transmiten. Con la fórmula de que se habla en el texto aplicada a algunas de las más comunes densidades obtenemos los valores siguientes:

Velocidad baudio	Valor
50	2690
110	1221
300	446
600	222
1200	110
2400	54
4800	28
9600	12
19200	5

El valor de la velocidad en baudios deseada se coloca (POKE) en la variable de sistema BAUD en las direcciones 23747 y 23748. El que exista una sola variable significa, por desgracia, que debe usarse la misma velocidad de transmisión para la emisión y para la recepción. El siguiente fragmento en código máquina ilustra el empleo de estos valores. Para establecer una velocidad de 300 baudios:

```
LD      HL,446
LD      (23747),HL
RET
```

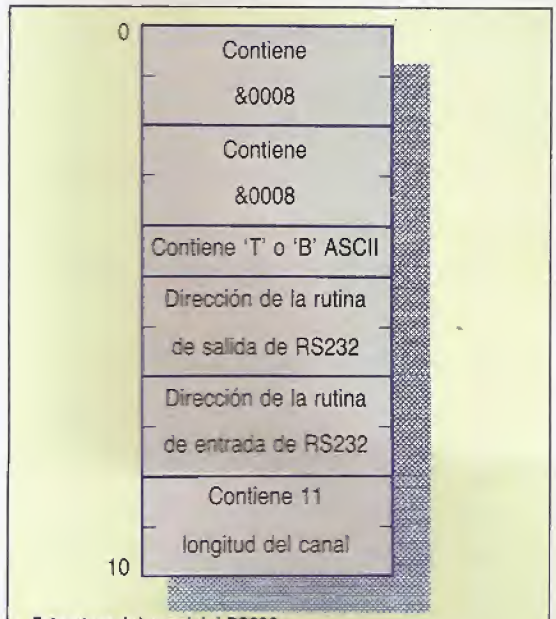
valor = (3500000 / (26 * velocidad en baudios))–2

Con esta ecuación es posible establecer la interface serial en una velocidad en baudios no habitual desde el código máquina, pero quizá no tenga utilidad alguna si usted no está comunicando con otro Spectrum a través de la RS232. La tabla de velocidades proporciona varios valores y la rutina necesaria para su implementación.

Veamos ahora el modo como son transferidos los datos a través del enlace. En BASIC, la apertura (OPEN) de una corriente a emplear con la interface RS232 genera un canal de la forma mostrada en el gráfico *Estructura del canal del RS232*. Este canal está situado en el área de canales de la memoria. El byte 4 de este canal tiene añadido 128 a su valor si el canal ha sido incrementado implícitamente (es decir, por medio de SAVE*, MOVE, FORMAT, etc.).

Pero esto sólo tiene un interés académico para nosotros cuando lo que vamos a emplear son las rutinas del RS232 desde el código máquina. Note que no existe ningún buffer asociado a tal canal. Escribimos y leemos datos llamando al correspondiente código de enganche, que a su vez llama a la rutina cuya dirección está o bien en las posiciones de canal 5 y 6 (transmisión) o bien en la 7 y 8 (recepción). No hay establecimiento de canal cuando se emplea un código de enganche. Los dos códigos de enganche del RS232 son:

- *Código de enganche 29:* Cuando se le llama da un único byte de la interface serial en el registro A si hay uno disponible (en este caso el flag de arrastre

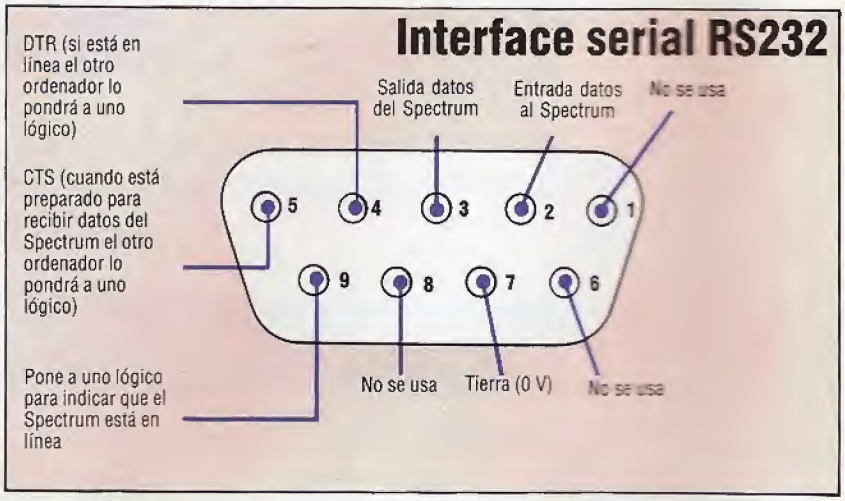


Estructura del canal del RS232
Observe que no hay ningún buffer asociado al canal. Todos los datos por enviar o recibir se pasan al registro A por medio de los códigos de enganche 29 y 30

Caroline Clayton

se pondrá a uno). Si no hay un byte disponible, A contendrá un 0 y el flag se pondrá a 0.

- *Código de enganche 30:* Transfiere el byte contenido en el registro A a la interface serial para su transmisión según la velocidad en baudios establecida. Al emplear este código, todos los bytes son transmitidos fielmente: no hay diferencia entre las corrientes B y T a nivel de lenguaje máquina.



Kevin Jones

Medios de comunicación
La puerta serial de la Interface 1 ofrece un medio de comunicación con otros dispositivos seriales y emplea un enchufe D de nueve patillas. La única desventaja de la interface serial del Spectrum es su incapacidad para manejar diferentes velocidades de transmisión en la emisión y en la recepción

Y es en este punto donde los problemas relacionados con SERBYT y SERFL pueden surgir. Si la rutina del código de enganche encuentra SERFL puesta a uno, tomará un byte de SERBYT, aun cuando ese byte haya sido «escondido» en SERBYT desde que se completó el último impulso de la comunicación serial. Este «byte polizón» exige ser atendido si aparece al comienzo de una serie de bytes recibidos: todos los bytes siguientes pueden ser recibidos, pero con un significado alterado. La manera más fácil de librarse de lo que pueda haber en SERBYT consiste simplemente en poner SERFL a cero antes de tratar de leer cualquier cosa desde la interface serial. Un sencillo código máquina lo hará:

```
XOR     A
LD      (23751),A
```

Ésta es la única complicación efectiva que mucha gente puede encontrar cuando emplea la interface serial, aunque si no puede usted sacar nada de ella, merece siempre la pena intercambiar las conexiones a las patillas 2 y 3 en el enchufe de la Interface 1, en caso de que hayan sido conectadas erróneamente en una primera instancia. La velocidad en baudios generada, aunque no siempre definida con precisión, es correcta dentro de la holgura tolerada por la especificación del RS232.

Como ejemplo de empleo de la interface RS232, el siguiente fragmento de código máquina transmite 255 bytes de datos por la interface serial:

```
                ;envia 255 bytes de datos por la interface serial
                ;
CF              rst     #8          establece...
31              defb    49          ...las variables de la I/face1
216E00          ld      hl,7278     valor para 1200 baudios
22C35C          ld      (23747),hl  establece veloc. en baudios
06FF            ld      b,255       establece contador
C5      loop:   push    bc          guarda contador
3E00            ld      a,23473     el usuario pone la rutina...
CF              rst     #8          ...para tomar datos del reg a
1E              defb    30          envia el byte contenido en a
C1              pop     bc          restaura contador
10F8            djnz    loop
C9              ret
```

Por muchos motivos la conexión RS232 es la más fácil de las funciones de la Interface 1 que se emplean. No hay que preocuparse por ninguna información del canal desde el código máquina, y sólo se dispone de dos llamadas —transmisión y recepción— que hay que atender.

Datos básicos (III)

Continuamos el análisis del mapa de memoria del C64, con información suministrada por Commodore Business Machines

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
FACHO	0062-0065	98-101	Acumulador #1 c. flotante: mantisa
FACSGN	0066	102	Acumulador #1 c. flotante: signo
SGNFLG	0067	103	Puntero: constante evaluación series
BITS	0068	104	Acumulador #1 c. flotante: dígito desbordamiento
ARGEXP	0069	105	Acumulador #2 c. flotante: expon.
ARGHO	006A-006D	106-109	Acumulador #2 c. flotante: mantisa
ARGSGN	006E	110	Acumulador #2 c. flotante: signo
ARISGN	006F	111	Resultado signo comparación: Acumul. #1 frente #2
FACOV	0070	112	Acumulador #1 c. flotante: orden inferior (redondeo)
FBUFPT	0071-0072	113-114	Puntero: buffer cassette
CHRGET	0073-008A	115-138	Subrutina: tomar byte siguiente del texto BASIC
CHRGOT	0079	121	Entrada a tomar mismo byte texto BASIC
TXTPTR	007A-007B	122-123	Puntero: byte actual texto BASIC
RNDX	008B-008F	139-143	Valor orig. función RND c. flotante
STATUS	0090	144	Palabra estado E/S Kernal: ST
STKEY	0091	145	Flag: tecla STOP/tecla RVS
SVXT	0092	146	Constante temporizado para cinta
VERCK	0093	147	Flag:0=carga, 1=verificación
C3PO	0094	148	Flag:Bus serial-carac. salida en buffer
BSOUR	0095	149	Carácter en buffer bus serial
SYNO	0096	150	Número sincr. cassette
	0097	151	Área datos temporal
LDTND	0098	152	Número archivos abiertos/tabla índice archivos